Available online at www.sciencedirect.com
ScienceDirect
The Journal of Logic and Algebraic Programming 73 (2007) 70–96
THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING
www.elsevier.com/locate/jlap

Reversing algebraic process calculi ☆

Iain Phillips [a,*], Irek Ulidowski [b]

[a] *Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2AZ, United Kingdom*
[b] *Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, United Kingdom*

Received 26 July 2006; accepted 27 November 2006
Available online 1 May 2007

Abstract

Reversible computation has a growing number of promising application areas such as the modelling of biochemical systems, program debugging and testing, and even programming languages for quantum computing. We formulate a procedure for converting operators of standard algebraic process calculi such as CCS into reversible operators, while preserving their operational semantics. We define forward–reverse bisimulation and show that it is preserved by all reversible operators.

Keywords: Reversible computation; Structural Operational Semantics (SOS); Formats of SOS rules; CCS with Keys; Forward–reverse bisimulation; Congruence result

1. Introduction

Reversible computation has a growing number of promising application areas such as the modelling of biochemical systems [10], program debugging and testing [29], and even programming languages for quantum computing [2]. Landauer [23] showed how irreversible computation generates heat; the efficient operation of future miniaturised computing devices could depend on exploiting reversibility [8]. We have been inspired to look at this area by the work of Danos and Krivine on reversible CCS [10–12] and Abramsky on mapping functional programs into reversible automata [1].

We wish to investigate reversibility for algebraic process calculi in the style of CCS [24], with Structural Operational Semantics (SOS) [28] rules. Given a forward *labelled transition relation* (ltr) $\rightarrow$ we are interested in obtaining a *reverse ltr* $\rightsquigarrow$ which is the inverse of $\rightarrow$. This can always be done, but if we just reverse a standard process language we end up with too many possibilities, since processes do not "remember" their past states. Danos and Krivine solve this problem by storing "memories" of past behaviour which are carried along with processes. Memories also keep track of which thread or threads performed an action. This has the effect that backtracking does not have to follow the exact order of

☆ An extended abstract of this work appeared at FOSSACS 2006 as [27].

* Corresponding author.
E-mail address: i.phillips@imperial.ac.uk (I. Phillips).

1567-8326/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2006.11.002

forward computation in reverse. To take a simple example, suppose that the process $a.b \mid c$ performs a followed by b and then c (here "." and "|" are the prefixing and the parallel composition of CCS, respectively). The process can backtrack by reversing b, then a and finally c. However, a cannot be reversed before b has been reversed.

We wish to produce reversible process calculi without relying on external devices such as memories. Our starting point is that irreversibility in a language such as CCS comes from the consumption of guards and alternative choices. We therefore decide to leave these in place, so that process structure remains fixed throughout a computation. Returning to the example of $a.b \mid c$, we might let the state after a, b and c have been performed be denoted by $\underline{a}.\underline{b} \mid \underline{c}$, where the underlined actions are *past* actions. There is plainly just the right amount of information here to reverse the process, while allowing $\underline{c}$ to be reversed independently of $\underline{b}$ and $\underline{a}$. This approach allows us also to keep track of unused alternatives discarded during computation. Consider $a.b + c$, where "+" is the choice operator of CCS. After the initial a, the alternative c is discarded and we can only proceed with b. This state is represented as $\underline{a}.b + c$; it is clear which alternative was taken and what will happen next.

It is interesting to note that reversibility can help to some extent to distinguish concurrency from causation. In the reversible world, Milner's expansion law does not hold: we have $a \mid b \neq a.b + b.a$ since $a \mid b \xrightarrow{a} \xrightarrow{b} \underline{a} \mid \underline{b} \overset{a}{\rightsquigarrow}$ but $a.b + b.a \xrightarrow{a} \xrightarrow{b} \underline{a}.\underline{b} + b.a \overset{a}{\not\rightsquigarrow}$.

When we come to consider autoconcurrency and communication we find that the simple method just outlined arguably discards too much information. For instance, the processes $a \mid a$ and $a.a$ cannot be distinguished by the above argument, as we are not able to tell apart the two occurrences of a. Moreover, the process $a \mid \overline{a}$ can evolve by a communication between a and the complementary action $\overline{a}$ to yield $\underline{a} \mid \underline{\overline{a}}$. This state could also have been reached by performing the two actions separately, and there is nothing in the notation to stop us from undoing the two actions separately. But if we are modelling biochemical systems and the communication represents a physical binding between two entities, then such separate backtracking of a and $\overline{a}$ is not reasonable.

As a further example, consider $(a.b \mid a.c \mid \overline{a}.d \mid \overline{a}.e) \setminus a$. Here the restriction $\setminus a$ prevents a and $\overline{a}$ being performed except as part of a communication. Suppose that $a.b$ communicates with $\overline{a}.d$ and $a.c$ with $\overline{a}.e$, giving $(\underline{a}.b \mid \underline{a}.c \mid \underline{\overline{a}}.d \mid \underline{\overline{a}}.e) \setminus a$. Then the notation allows us to backtrack by undoing a different pair of actions to get $(a.b \mid \underline{a}.c \mid \underline{\overline{a}}.d \mid \overline{a}.e) \setminus a$, which is counterintuitive if undoing a communication corresponds to unbinding two entities.

Danos and Krivine handle this issue using threads. When a communication occurs, it is between two threads, and these threads are locked together by giving each thread the name of the other thread, which is added to each thread's memory.

Our solution is to use a more expressive form of past actions, where each occurrence of action a is "marked" by a fresh identifier m and written as $a[m]$. Also, we insist that the two parties to a communication between action a and $\overline{a}$ agree on this identifier or *communication key*, which is unique to that communication. This means that a and $\overline{a}$ are now locked together and can only be undone together. Now, we can deal with the autoconcurrency example. Process $a \mid a$ can perform the a actions with keys m and n to produce $a[m] \mid a[n]$, and then reverse these actions in any order. However, $a.a$ cannot match this behaviour: after the a actions with keys m and n, the process $a[m].a[n]$ cannot reverse on $a[m]$. We can also handle communication correctly. For instance, it is clear that $a[m] \mid \overline{a}[m]$ must have been produced by a communication, whereas $a[m] \mid \overline{a}[n]$ $(m \neq n)$ comes from two separate transitions.

We propose a method for reversing process operators that are definable by SOS rules in a general format. As far as we are aware, this is the first time this has been done for algebraic process calculi. As we have described informally above, we rely on reformulating operators of standard process calculi into new operators that can be easily reversed, while preserving their operational meaning. In this paper we attempt to balance the generality of the format on one hand and the technical simplicity of the proposed method on the other hand. The chosen format is general enough for the definitions of the majority of useful process operators, and the method presented is intuitive and easy to apply.

Our format is a subformat of the *path* format [3] and consists of *dynamic* rules, where the operator is destroyed by a transition, and *static* rules, where the operator remains present after the transition. Reversing static rules is easier because they preserve the context during execution. Dynamic rules, however, consume the context, removing the unused alternatives. The kernel of our method is to transform dynamic rules into static-like rules. The method is summarised as follows:

(1) Past behaviour and discarded alternatives are recorded in the *syntax of terms* and not on external devices such as memories. This is achieved by reformulating operators, where necessary, to equivalent *static* versions.

(2) *Auxiliary operators* and *predicates* are used to keep the structure of terms unchanged and to enforce correct use of subterms in the reformulated contexts.
(3) *Symmetry* between forward and reverse SOS rules. Once SOS rules for operators are reformulated as above, the reverse SOS rules are obtained simply as symmetric versions of the forward rules.

As an illustration of the method we consider the CCS choice operator $+$. We reformulate it as a static operator and use predicate std, meaning that the argument is a standard term that uses no past actions (and no keys), to control when arguments can fire in rules.

$$\frac{X \xrightarrow{a} X' \quad \mathsf{std}(Y)}{X + Y \xrightarrow{a} X' + Y} \qquad \frac{Y \xrightarrow{a} Y' \quad \mathsf{std}(X)}{X + Y \xrightarrow{a} X + Y'}$$

The reverse rules for the converted $+$ are then obtained by symmetry:

$$\frac{X \overset{a}{\rightsquigarrow} X' \quad \mathsf{std}(Y)}{X + Y \overset{a}{\rightsquigarrow} X' + Y} \qquad \frac{Y \overset{a}{\rightsquigarrow} Y' \quad \mathsf{std}(X)}{X + Y \overset{a}{\rightsquigarrow} X + Y'}$$

We prove a number of results to show that our method yields well-behaved transition relations. We show that the new forward ltr is conservative over the standard ltr (Theorem 5.21). Also the new forward and reverse ltrs satisfy certain confluence properties (Propositions 5.10 and 5.19). The processes which are reachable from standard processes by forward-only transitions are closed under reverse transitions, meaning that a process can never reverse into an "inconsistent" past (Proposition 5.15). We also formulate a notion of *forward–reverse bisimulation*, which is a congruence for operators whose original forward rules belong to our format (Theorem 6.7).

The rest of the paper is structured as follows. In Section 2 we define the simple process calculi which we shall be making reversible, and in Section 3 we describe our procedure for generating the new reversible calculi. In Section 4 we illustrate our method by applying it to CCS. We also discuss related work, and in particular RCCS [11]. In Section 5 we prove various results about the new reversible transition relations, and in Section 6 we define an appropriate notion of bisimulation. Section 7 indicates how to adapt the method to a more general format that contains action constants, predicates and recursion. We end with some conclusions.

2. Process calculi

In this section we describe the process calculi to which we shall apply our procedure for generating reversible calculi.

A *signature* is a set Σ of operator symbols, each with a particular arity. Given a set of variables V, ranged over by $X, Y, \ldots$, the set of (*open*) *terms* over Σ is denoted by $T(\Sigma, V)$, ranged over by $t, \ldots$. We may write a term t as $t(X_1, \ldots, X_n)$ to indicate that $t \in T(\Sigma, \{X_1, \ldots, X_n\})$. We abbreviate $T(\Sigma, \emptyset)$ by $T(\Sigma)$; this is the set of *closed* terms over Σ. We shall tend to refer to closed terms as *processes*. We let $P, Q, \ldots$ range over processes.

A *process calculus* $L = (\Sigma, A, R)$, is given by a signature Σ, a set of actions A and a set R of SOS rules. We shall apply our procedure to a "standard" calculus $L_{\mathrm{S}} = (\Sigma_{\mathrm{S}}, \mathsf{Act}, R_{\mathrm{S}})$. Its terms are called *standard* terms and are denoted by Std. We shall assume that the only operator of arity zero (i.e. constant) is the deadlocked process $\mathbf{0}$. We let $f, \ldots$ range over Σ_{S}; $a, b, c, \ldots$ range over Act.

We next describe the rules R which define the operational semantics of terms.

The SOS theory gives us the flexibility and the benefits of working with whole classes of process calculi rather than with individual process calculi that are limited to a small number of operators. Typically, a class of operators is defined by a format of SOS rules that can be used to define them operationally. In this paper we shall consider simple *path* rules without copying [3]. More specifically, our rules will be mostly of the simpler *pxyft* and *pxf* forms, where terms in the premises are variables and the source of the conclusion is a term constructed with a single operator.

Definition 2.1. *Simple path* (forward) rules are expressions of the form

$$\frac{\{\, X_i \xrightarrow{a_i} X_i' \,\}_{i \in I} \quad \{\, \mathsf{p_j}(X_j) \,\}_{j \in J}}{f(X_1, \ldots, X_n) \xrightarrow{a} t(X_1', \ldots, X_n')} \qquad \text{and} \qquad \frac{\{\, \mathsf{p_j}(X_j) \,\}_{j \in J}}{\mathsf{p}(f(X_1, \ldots, X_n))},$$

where $I, J \subseteq \{1, \ldots, n\}$, all variables X_i (X_j) and X_i' are distinct, and variables X_i' are such that $X_i' = X_i$ when $i \notin I$.

The sets of transitions and predicate expressions above the horizontal bars in the rules above are called *premises*. Let r be the first rule above. Operator f is the *operator* of r. The transition below the bar in r is the *conclusion* of r. Action a in the conclusion is the *action* of r and $f(X_1, \ldots, X_n)$ and $t(X'_1, \ldots, X'_n)$ are the *source* and *target* of r, respectively. The ith argument is *active* in r if r has a transition for X_i in the premises. The ith argument of f is *active* if it is active in some rule for f. In the second rule, p is the *predicate* of the rule and the predicate expression below the bar is the *conclusion*.

With any calculus $L = (\Sigma, A, R)$, all of whose rules are in simple *path* format, we associate an ltr $\rightarrow$ with labels in A, together with a set of predicates, in the standard way; for details see [3]. Our standard calculus L_{S} will have all its rules R_{S} in simple *path* format. However, it will have no predicates in its rules since, as we shall argue in Section 7.1, *path* rules with arbitrary predicates may give rise to transition relations that fail to satisfy a number of vital properties. We shall write the ltr for L_{S} as $\rightarrow_{\mathrm{S}}$, and use this in writing down its rules for clarity.

We now define the precise form of SOS rules that operators of L_{S} can have. Consider an n-ary operator $f \in \Sigma_{\mathrm{S}}$ ($n \geq 1$). The set of arguments of f is $N_f = \{1, \ldots, n\}$. Operator f can have three kinds of rules: static rules, choice rules and choice axioms. We describe each in turn. We abbreviate $X_1, \ldots, X_n$ by $\overrightarrow{X}$.

Definition 2.2. *Static rules* of f are of the following form, where $I \neq \emptyset$ and $X'_i = X_i$ for all $i \notin I$:

$$\text{(I)} \quad \frac{\{X_i \xrightarrow{a_i}_{\mathrm{S}} X'_i\}_{i \in I}}{f(\overrightarrow{X}) \xrightarrow{a}_{\mathrm{S}} f(\overrightarrow{X'})}$$

We require that if two static rules for f have the same premises then they have the same conclusion (i.e. the action of the conclusion is unique). Let $S_f \subseteq N_f$ be the set of all arguments occurring in the premises of static rules of f, and let $E_f = N_f \setminus S_f$. Arguments in S_f are called *static arguments*.

As an example, consider a CSP-style [22] parallel composition operator with rule schemas as follows:

$$\frac{X \xrightarrow{a}_{\mathrm{S}} X' \quad Y \xrightarrow{a}_{\mathrm{S}} Y'}{X||_A Y \xrightarrow{a}_{\mathrm{S}} X'||_A Y'} \quad (a \in A)$$

$$\frac{X \xrightarrow{b}_{\mathrm{S}} X'}{X||_A Y \xrightarrow{b}_{\mathrm{S}} X'||_A Y} \quad (b \notin A) \qquad \frac{Y \xrightarrow{b}_{\mathrm{S}} Y'}{X||_A Y \xrightarrow{b}_{\mathrm{S}} X||_A Y'} \quad (b \notin A)$$

Here $A \subseteq \mathsf{Act}$ is a set of actions on which processes are required to synchronise. Both arguments are static, so that $S_{||_A} = \{X, Y\}$ and $E_{||_A} = \emptyset$.

The arguments of the CCS parallel composition operator are static, as are those of the CCS restriction and relabelling operators and the CSP hiding operator. Also, the rules for the first argument of Milner's interrupt operator "$\hat{}$" at the end of this section are static. Note that, of course, the order of arguments of the source of (I) is the same as that of the target. We show at the end of the section that static-like operators that "swap" their arguments can be reversed but lead to processes with undesirable properties.

To see why we need the condition that if two static rules for f have the same premises then they have the same conclusion, consider an example operator f with just the following two static rules:

$$\frac{X \xrightarrow{a}_{\mathrm{S}} X'}{f(X) \xrightarrow{b}_{\mathrm{S}} f(X')} \qquad \frac{X \xrightarrow{a}_{\mathrm{S}} X'}{f(X) \xrightarrow{c}_{\mathrm{S}} f(X')}$$

Suppose that we have a term $f(P')$, such that P' can be reversed by performing a to reach P. Then using the two rules for f, we see that $f(P')$ can reverse by performing either b or c. So operator f introduces an undesirable ambiguity when reversed. We shall establish a reverse confluence property (Proposition 5.10) for our format of operators, which fails in this example, since the actions b and c are in conflict.

Next we describe the choice rules.

Definition 2.3. A *choice rule* of f is a rule of the following form:

$$\text{(II)} \quad \frac{X_d \xrightarrow{a}_{\mathrm{S}} X'_d}{f(\overrightarrow{X}) \xrightarrow{a}_{\mathrm{S}} X'_d}$$

We require that $d \in E_f$. Let D_f be the set of all arguments d occurring in the premises of choice rules of f. Arguments in D_f are called *dynamic* arguments. Each dynamic argument d is required to be *permissive*, meaning that for each $a \in \mathsf{Act}$ there is a rule of type (II).

Note that $D_f \subseteq E_f$, so that a dynamic argument cannot be static.

The choice operator of CCS has two dynamic arguments, both of which are permissive. The second argument of Milner's interrupt operator is dynamic and permissive. The external choice operator of CSP also has two dynamic arguments, but they are not permissive: although they have choice rules for all $a \in \mathsf{Act} \setminus \{\tau\}$, they have no such rules for the τ—the rules for τ are static. We discuss CSP external choice further in Section 7.1.

We also wish to encompass operators that have choice rules with empty premises such as, for example, CCS prefixing and CSP internal choice. This leads us to the third and final type of rule:

Definition 2.4. A *choice axiom* of f is a rule r of the following form:

$$\text{(III)} \quad r \ \frac{}{f(\overrightarrow{X}) \xrightarrow{a}_{\mathrm{S}} X_{\mathsf{ta}(r)}}$$

Here $\mathsf{ta}(r)$ is the *target argument*. We require $\mathsf{ta}(r) \in E_f$.

Next, we define the class of simple process calculi that we shall reverse.

Definition 2.5. A process operator f is *simple* if either f is the deadlocked process $\mathbf{0}$, or f has a nonzero arity and all its rules are as in Definitions 2.2, 2.3 and 2.4.

In what follows we omit the subscripts of S_f, E_f and D_f where no confusion can arise.

We shall require that L_{S} is simple. The main application of this work is to reformulate and reverse Milner's CCS, and many other operators from the process calculi ACP [4] and CSP [22] and their descendants.

3. The procedure for generating a reversible calculus

We shall transform L_{S} into an operationally equivalent calculus which is easily reversible. For this we shall need to augment the processes and reformulate the rules of L_{S}.

Let $\mathcal{K}$ be an infinite set of *communication keys* (or just *keys* for short), ranged over by $m, n, \ldots$. The set of *past actions*, or actions marked with keys, is denoted by $\mathsf{ActK} = \mathsf{Act} \times \mathcal{K}$. We write the ordered pair (a, m) as $a[m]$. We let $\mu, \nu, \ldots$ range over ActK, and $s, t, \ldots$ range over ActK^*.

We introduce the signature Σ_{A} of auxiliary operators $f_r[m]$, where r is a rule of type (III) for an operator f of Σ_{S}, and $m \in \mathcal{K}$. We let $\Sigma_{\mathrm{SA}} = \Sigma_{\mathrm{S}} \cup \Sigma_{\mathrm{A}}$, and let $\mathsf{Proc} = T(\Sigma_{\mathrm{SA}})$. Clearly, $\mathsf{Std} \subseteq \mathsf{Proc}$.

Our reformulation and reversing method relies on auxiliary unary predicates on Proc, namely $\mathsf{std}(P)$ and $\mathsf{fsh}[m](P)$ (all $m \in \mathcal{K}$). Informally, $\mathsf{std}(P)$ holds if $P \in \mathsf{Std}$ and $\mathsf{fsh}[m](P)$ holds if key m is fresh (i.e. not used) in P. The predicates are defined below, where the last four rules are rule schemas for all relevant operators and keys

$$\frac{}{\mathsf{std}(\mathbf{0})} \qquad \frac{\{\mathsf{std}(X_i)\}_{i\in N}}{\mathsf{std}(f(\overrightarrow{X}))}$$

$$\frac{}{\mathsf{fsh}[m](\mathbf{0})} \qquad \frac{\{\mathsf{fsh}[m](X_i)\}_{i\in N}}{\mathsf{fsh}[m](f(\overrightarrow{X}))} \qquad \frac{\{\mathsf{fsh}[m](X_i)\}_{i\in N}}{\mathsf{fsh}[m](f_r[n](\overrightarrow{X}))}\ m \neq n$$

Note that if $\mathsf{std}(P)$ then $\mathsf{fsh}[m](P)$ for every $m \in \mathcal{K}$. Let R_{P} be the set of rules for the predicates std and $\mathsf{fsh}[m]$ for all $m \in \mathcal{K}$.

We define how to transform rules of type (I)–(III) into rules in simple *path* format that can be easily reversed.

Definition 3.1. For every operator f in Σ_S, every static rule of type (I) for f is converted into

$$(1)\quad \frac{\{X_i \overset{a_i[m]}{\to} X_i'\}_{i\in I} \quad \{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S\setminus I}}{f(\overrightarrow{X}) \overset{a[m]}{\to} f(\overrightarrow{X'})}$$

where $X_i' = X_i$ for all $i \notin I$. The reverse version is

$$(1\mathrm{R})\quad \frac{\{X_i \overset{a_i[m]}{\rightsquigarrow} X_i'\}_{i\in I} \quad \{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S\setminus I}}{f(\overrightarrow{X}) \overset{a[m]}{\rightsquigarrow} f(\overrightarrow{X'})}$$

Note that (1) and (1R) are rule schemas for keys m. Also, $I \cap E = \emptyset$, and so predicates only apply to inactive arguments. This contributes to making our rules easily reversible. Finally note that we shall be able to prove that if $P \overset{a[m]}{\to} P'$ then $\mathsf{fsh}[m](P)$ (Lemma 5.2).

Example 3.2. The interleaving operator $|||$ of CSP [21] has the following forward rules:

$$\frac{X \overset{a}{\to}_S X'}{X \mid\mid\mid Y \overset{a}{\to}_S X' \mid\mid\mid Y} \qquad \frac{Y \overset{a}{\to}_S Y'}{X \mid\mid\mid Y \overset{a}{\to}_S X \mid\mid\mid Y'}$$

Both arguments of $|||$ are static and no argument is dynamic. Hence, we have $S = N = \{X, Y\}$ and $E = \emptyset$. By Definition 3.1, these rules are converted to the two rules below. Since $E = \emptyset$, predicates $\mathsf{std}()$ do not appear in the rules.

$$\frac{X \overset{a[m]}{\to} X' \quad \mathsf{fsh}[m](Y)}{X \mid\mid\mid Y \overset{a[m]}{\to} X' \mid\mid\mid Y} \qquad \frac{Y \overset{a[m]}{\to} Y' \quad \mathsf{fsh}[m](X)}{X \mid\mid\mid Y \overset{a[m]}{\to} X \mid\mid\mid Y'}$$

Reverse rules are obtained by simply taking symmetric versions of the forward rules:

$$\frac{X \overset{a[m]}{\rightsquigarrow} X' \quad \mathsf{fsh}[m](Y)}{X \mid\mid\mid Y \overset{a[m]}{\rightsquigarrow} X' \mid\mid\mid Y} \qquad \frac{Y \overset{a[m]}{\rightsquigarrow} Y' \quad \mathsf{fsh}[m](X)}{X \mid\mid\mid Y \overset{a[m]}{\rightsquigarrow} X \mid\mid\mid Y'}$$

Definition 3.3. For every operator f in Σ_S, every choice rule of type (II) for f is converted into

$$(2)\quad \frac{X_d \overset{a[m]}{\to} X_d' \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{d\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f(\overrightarrow{X}) \overset{a[m]}{\to} f(\overrightarrow{X'})}$$

where $X_i' = X_i$ for all $i \neq d$. The reverse version of (2) is

$$(2\mathrm{R})\quad \frac{X_d \overset{a[m]}{\rightsquigarrow} X_d' \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{d\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f(\overrightarrow{X}) \overset{a[m]}{\rightsquigarrow} f(\overrightarrow{X'})}$$

Again (2) and (2R) are rule schemas for keys m, and again predicates are only applied to inactive arguments, since $d \notin S$.

Example 3.4. The nondeterministic choice operator of CCS has the standard choice rules, where both arguments are dynamic, $E = D = \{X, Y\}$ and $S = \emptyset$:

$$\frac{X \overset{a}{\to}_S X'}{X + Y \overset{a}{\to}_S X'} \qquad \frac{Y \overset{a}{\to}_S Y'}{X + Y \overset{a}{\to}_S Y'}$$

Clearly, both arguments of $+$ are permissive. By Definition 3.3, the rules get converted to the two rules below. Since $S = \emptyset$, predicates $\mathsf{fsh}[m]()$ do not feature in the rules

$$\frac{X \overset{a[m]}{\to} X' \quad \mathsf{std}(Y)}{X + Y \overset{a[m]}{\to} X' + Y} \qquad \frac{Y \overset{a[m]}{\to} Y' \quad \mathsf{std}(X)}{X + Y \overset{a[m]}{\to} X + Y'}$$

Reverse rules are the symmetric versions of the converted rules:

$$\frac{X \overset{a[m]}{\rightsquigarrow} X' \quad \mathsf{std}(Y)}{X+Y \overset{a[m]}{\rightsquigarrow} X'+Y} \qquad \frac{Y \overset{a[m]}{\rightsquigarrow} Y' \quad \mathsf{std}(X)}{X+Y \overset{a[m]}{\rightsquigarrow} X+Y'}$$

In order to handle operators f with rules of type (III), we shall use auxiliary operators. These operators have their own rules (type (3′) below) which propagate the actions of a single argument leaving other arguments unchanged.

Definition 3.5. For every operator f in Σ_S, every rule r of type (III) for f is converted into the rule schemas below for all $b \in \mathsf{Act}$ and keys m, n:

$$(3) \quad \frac{\{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f(\overrightarrow{X}) \overset{a[m]}{\rightarrow} f_r[m](\overrightarrow{X})}$$

$$(3') \quad \frac{X_{\mathsf{ta}(r)} \overset{b[m]}{\rightarrow} X'_{\mathsf{ta}(r)} \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{\mathsf{ta}(r)\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f_r[n](\overrightarrow{X}) \overset{b[m]}{\rightarrow} f_r[n](\overrightarrow{X'})}\ m \neq n$$

The reverse versions of rule schemas of type (3) and (3′) are

$$(3\mathrm{R}) \quad \frac{\{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f_r[m](\overrightarrow{X}) \overset{a[m]}{\rightsquigarrow} f(\overrightarrow{X})}$$

$$(3'\mathrm{R}) \quad \frac{X_{\mathsf{ta}(r)} \overset{b[m]}{\rightsquigarrow} X'_{\mathsf{ta}(r)} \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{\mathsf{ta}(r)\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f_r[n](\overrightarrow{X}) \overset{b[m]}{\rightsquigarrow} f_r[n](\overrightarrow{X'})}\ m \neq n$$

Again predicates are only applied to inactive arguments.

Notice that rules of type (3′) are static in nature, while those of type (3) are dynamic in nature (the operator changes in the conclusion).

Example 3.6. We illustrate how to convert operators with SOS rules of type (III) by considering the action prefixing of CCS and the internal choice of CSP.

CCS prefixing is defined by the following standard rules (one for each action a belonging to a set Act):

$$\frac{}{a.X \overset{a}{\rightarrow}_S X}$$

Thus we have a family of operators, one for each $a \in \mathsf{Act}$.

By Definition 3.5 we introduce auxiliary operators $a[m].X$ and get the following forward and reverse rules:

$$\frac{\mathsf{std}(X)}{a.X \overset{a[m]}{\rightarrow} a[m].X} \qquad \frac{X \overset{b[n]}{\rightarrow} X'}{a[m].X \overset{b[n]}{\rightarrow} a[m].X'}\ m \neq n$$

$$\frac{\mathsf{std}(X)}{a[m].X \overset{a[m]}{\rightsquigarrow} a.X} \qquad \frac{X \overset{b[n]}{\rightsquigarrow} X'}{a[m].X \overset{b[n]}{\rightsquigarrow} a[m].X'}\ m \neq n$$

The internal choice operator $\sqcap$ of CSP may be defined by two choice axioms ($\tau \in \mathsf{Act}$):

$$\frac{}{X \sqcap Y \overset{\tau}{\rightarrow}_S X} \qquad \frac{}{X \sqcap Y \overset{\tau}{\rightarrow}_S Y}$$

Arguments X and Y both belong to E. Definition 3.5 requires two families of auxiliary operators $\sqcap_1[m]$ and $\sqcap_2[m]$ for all $m \in \mathcal{K}$. Since the rules are symmetric, we only give the converted rules and the reverse rules for the first argument X:

$$\frac{\mathsf{std}(X) \quad \mathsf{std}(Y)}{X \sqcap Y \overset{\tau[m]}{\to} X \sqcap_1 [m]\, Y} \qquad \frac{X \overset{a[n]}{\to} X' \quad \mathsf{std}(Y)}{X \sqcap_1 [m]\, Y \overset{a[n]}{\to} X' \sqcap_1 [m]\, Y} \quad m \neq n$$

$$\frac{\mathsf{std}(X) \quad \mathsf{std}(Y)}{X \sqcap_1 [m]\, Y \overset{\tau[m]}{\rightsquigarrow} X \sqcap Y} \qquad \frac{X \overset{a[n]}{\rightsquigarrow} X' \quad \mathsf{std}(Y)}{X \sqcap_1 [m]\, Y \overset{a[n]}{\rightsquigarrow} X' \sqcap_1 [m]\, Y} \quad m \neq n$$

In order to see why we require $\mathsf{ta}(r) \in E$ in Definition 2.4, consider operator f defined by

$$\frac{}{f(X) \overset{c}{\to}_{\mathrm{S}} X} \qquad \frac{X \overset{a}{\to}_{\mathrm{S}} X'}{f(X) \overset{a}{\to}_{\mathrm{S}} f(X')}$$

Here we have a choice axiom and a static rule which share the same argument. If we transform the choice axiom to the rules following according to Definition 3.5, then we have "lost" the computation $f(a.\mathbf{0}) \overset{a}{\to}_{\mathrm{S}} f(\mathbf{0}) \overset{c}{\to}_{\mathrm{S}} \mathbf{0}$, since we get $f(a) \overset{a[n]}{\to} f(a[n])$ by the transformed static rule for f, and $f(a[n]) \overset{c[m]}{\not\to}$, because $a[n]$ is not a standard term.

$$\frac{\mathsf{std}(X) \quad \mathsf{fsh}[m](X)}{f(X) \overset{c[m]}{\to} f_r[m](X)} \qquad \frac{X \overset{b[n]}{\to} X' \quad \mathsf{fsh}[n](X)}{f_r[m](X) \overset{b[n]}{\to} f_r[m](X')} \quad m \neq n$$

Now we are ready to define our procedure that reformulates standard operators and produces automatically their new forward and reverse rules. Note that all rules mentioned in Definitions 3.1, 3.3 and 3.5 are in the simple *path* format.

Definition 3.7 (*Conversion Procedure*). A simple process calculus $L_{\mathrm{S}} = (\Sigma_{\mathrm{S}}, \mathsf{Act}, R_{\mathrm{S}})$ generates a *reversible process calculus with communication keys* $L = (\Sigma_{\mathrm{SA}}, \mathsf{ActK}, R_{\mathrm{F}}, R_{\mathrm{R}})$ as follows:

(1) $\Sigma_{\mathrm{SA}} \overset{\mathrm{def}}{=} \Sigma_{\mathrm{S}} \cup \Sigma_{\mathrm{A}}$. The operators in Σ_{SA} are called *reversible* operators.
(2) The forward rule set R_{F} is the least set such that
  (a) $R_{\mathrm{P}} \subseteq R_{\mathrm{F}}$, where R_{P} is the set of rules for predicates defined above;
  (b) for every rule $r \in R_{\mathrm{S}}$ for f of type (I) or (II) the set R_{F} contains the converted rules r' of the corresponding type (1) or (2) as required by Definitions 3.1 and 3.3;
  (c) for every rule $r \in R_{\mathrm{S}}$ for f of type (III) the set R_{F} contains the converted rule r' of type (3), and all the rules of type (3′) for the auxiliary operators $f_r[m]$ as required by Definition 3.5.
(3) The reverse rule set R_{R} is defined like R_{F}, except that we use the reverse forms of the rules as in Definitions 3.1, 3.3 and 3.5.

Once L is generated by the procedure in Definition 3.7, we associate with L, in the standard way [3], the forward and reverse ltrs $\to$ and $\rightsquigarrow$ over Proc with labels drawn from ActK, together with the set of predicates Pred that interpret std and $\mathsf{fsh}[m]$ (for $m \in \mathcal{K}$) over Proc.

We illustrate the application of the conversion procedure on an operator that has both static and dynamic rules. The operator is Milner's interrupt operator "^" [24] defined by the two rule schemas below (all $a, b \in \mathsf{Act}$).

$$\frac{X \overset{a}{\to}_{\mathrm{S}} X'}{X \hat{} Y \overset{a}{\to}_{\mathrm{S}} X' \hat{} Y} \qquad \frac{Y \overset{b}{\to}_{\mathrm{S}} Y'}{X \hat{} Y \overset{b}{\to}_{\mathrm{S}} Y'}$$

We have $S = \{X\}$, $D = \{Y\}$, $E = D$ and Y is permissive. Definitions 3.1 and 3.3 give us the two forward rule schemas below, and the reverse rules are simply symmetric versions of the forward rules.

$$\frac{X \overset{a[m]}{\to} X' \quad \mathsf{std}(Y)}{X \hat{} Y \overset{a[m]}{\to} X' \hat{} Y} \qquad \frac{Y \overset{b[n]}{\to} Y' \quad \mathsf{fsh}[n](X)}{X \hat{} Y \overset{b[n]}{\to} X \hat{} Y'}$$

$$\frac{X \overset{a[m]}{\rightsquigarrow} X' \quad \mathsf{std}(Y)}{X \hat{} Y \overset{a[m]}{\rightsquigarrow} X' \hat{} Y} \qquad \frac{Y \overset{b[n]}{\rightsquigarrow} Y' \quad \mathsf{fsh}[n](X)}{X \hat{} Y \overset{b[n]}{\rightsquigarrow} X \hat{} Y'}$$

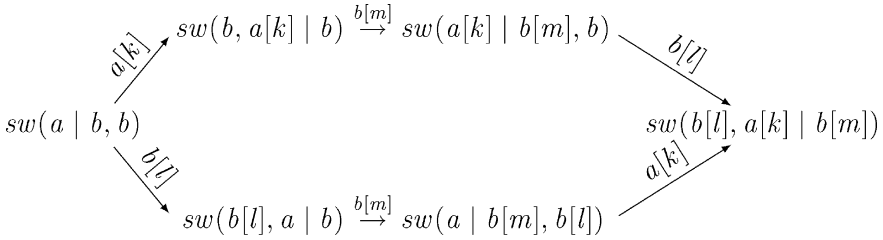

Fig. 1. The swap operator.

We conclude this section with an example of operator "sw" that swaps its two arguments around every time a certain action is performed. Its standard SOS rules are as follows, where a and b are fixed actions (i.e. these are not rule schemas):

$$\frac{X \overset{a}{\rightarrow}_{\mathrm{S}} X'}{sw(X,Y) \overset{a}{\rightarrow}_{\mathrm{S}} sw(Y,X')} \qquad \frac{Y \overset{b}{\rightarrow}_{\mathrm{S}} Y'}{sw(X,Y) \overset{b}{\rightarrow}_{\mathrm{S}} sw(Y',X)}$$

The rules of this operator satisfy almost all conditions required for static rules as in Definition 2.2. They only fail the definition because they swap the arguments X and Y. Operators like sw can be reversed by adapting our method. However, processes constructed with such operators do not satisfy some vital properties that we shall introduce in Section 5. The forward and reverse rules for sw are

$$\frac{X \overset{a[k]}{\rightarrow} X' \quad \mathsf{fsh}[k](Y)}{sw(X,Y) \overset{a[k]}{\rightarrow} sw(Y,X')} \qquad \frac{Y \overset{b[n]}{\rightarrow} Y' \quad \mathsf{fsh}[n](X)}{sw(X,Y) \overset{b[n]}{\rightarrow} sw(Y',X)}$$

$$\frac{Y \overset{a[k]}{\rightsquigarrow} Y' \quad \mathsf{fsh}[k](X)}{sw(X,Y) \overset{a[k]}{\rightsquigarrow} sw(Y',X)} \qquad \frac{X \overset{b[n]}{\rightsquigarrow} X' \quad \mathsf{fsh}[n](Y)}{sw(X,Y) \overset{b[n]}{\rightsquigarrow} sw(Y,X')}$$

where a and b are fixed and k and n are arbitrary keys. Consider a term $sw(a \mid b, b)$. A fragment of its derivation diagram is displayed in Fig. 1. Since $sw(b[l], a \mid b) \overset{a[k]}{\not\rightarrow}$ for any key k, we deduce that the Forward Diamond Property (Proposition 5.19) does not hold. Moreover, as $sw(a[k] \mid b[m], b) \overset{a[k]}{\not\rightsquigarrow}$ we know that the Reverse Diamond Property (Proposition 5.10) is not valid.

4. CCS with communication keys

In this section we convert CCS to a reversible process calculus, which we call CCSK (CCS with communication Keys), following Definition 3.7.

We define the actions of CCS as usual: let $\mathcal{A}$ be a set of names, ranged over by $\alpha, \ldots$, let $\overline{\alpha}$ be the *complement* of $\alpha \in \mathcal{A}$, let $\overline{\mathcal{A}} = \{\overline{\alpha} : \alpha \in \mathcal{A}\}$, and let Act be the disjoint union of $\mathcal{A}$, $\overline{\mathcal{A}}$ and $\{\tau\}$. Also, let $\overline{\overline{\alpha}} = \alpha$ for $\alpha \in \mathcal{A}$, and let $\overline{A} = \{\overline{a} : a \in A\}$ for $A \subseteq \mathsf{Act} \setminus \{\tau\}$.

We assume the following standard signature of finite CCS:

$$\Sigma_{\mathrm{S}} = \{\mathbf{0}\} \cup \{a. \mid a \in \mathsf{Act}\} \cup \{\backslash A, [f] \mid A \subseteq \mathsf{Act} \setminus \{\tau\}, f : \mathsf{Act} \to \mathsf{Act}\} \cup \{+, |\}$$

The single argument of prefixing is neither dynamic nor static, and prefixing has a choice axiom rule (type (III)). By Definition 3.5, CCSK contains a family of auxiliary operators $a[m].$ (past action prefixing) for all $a \in \mathsf{Act}$ and $m \in \mathcal{K}$. Both arguments of $+$ are dynamic and permissive, and obviously non-static. Parallel composition, restriction and relabelling are operators with static rules. The well-known SOS rules for CCS, which can be found in [24], are converted into the rules in Fig. 2. The rules for the reverse ltr for CCSK are got by simply changing $\rightarrow$ into $\rightsquigarrow$ throughout. As is usual, we omit trailing $\mathbf{0}$s. The reverse rules are displayed in Fig. 3.

Example 4.1. In CCSK we keep track of the identities of actions that communicate so that when we reverse we undo the correct past actions. Consider $P = (a.b \mid a.c \mid \overline{a}.d \mid \overline{a}.e)\backslash a$. Here the restriction of a prevents a and $\overline{a}$ being

$$\frac{\mathsf{std}\,(X)}{a.X \overset{a[m]}{\rightarrow} a[m].X} \qquad \frac{X \overset{b[n]}{\rightarrow} X'}{a[m].X \overset{b[n]}{\rightarrow} a[m].X'}\ m \neq n$$

$$\frac{X \overset{a[m]}{\rightarrow} X' \quad \mathsf{std}\,(Y)}{X + Y \overset{a[m]}{\rightarrow} X' + Y} \qquad \frac{Y \overset{a[m]}{\rightarrow} Y' \quad \mathsf{std}\,(X)}{X + Y \overset{a[m]}{\rightarrow} X + Y'}$$

$$\frac{X \overset{a[m]}{\rightarrow} X' \quad \mathsf{fsh}\,[m](Y)}{X \mid Y \overset{a[m]}{\rightarrow} X' \mid Y} \qquad \frac{Y \overset{a[m]}{\rightarrow} Y' \quad \mathsf{fsh}\,[m](X)}{X \mid Y \overset{a[m]}{\rightarrow} X \mid Y'}$$

$$\frac{X \overset{a[m]}{\rightarrow} X' \quad Y \overset{\overline{a}[m]}{\rightarrow} Y'}{X \mid Y \overset{\tau[m]}{\rightarrow} X' \mid Y'}\ (a \neq \tau)$$

$$\frac{X \overset{a[m]}{\rightarrow} X'}{X \backslash A \overset{a[m]}{\rightarrow} X' \backslash A}\ a \notin A \cup \overline{A} \qquad \frac{X \overset{a[m]}{\rightarrow} X'}{X[f] \overset{f(a)[m]}{\rightarrow} X'[f]}$$

Fig. 2. Forward SOS rules for CCSK.

$$\frac{\mathsf{std}\,(X)}{a[m].X \overset{a[m]}{\rightsquigarrow} a.X} \qquad \frac{X \overset{b[n]}{\rightsquigarrow} X'}{a[m].X \overset{b[n]}{\rightsquigarrow} a[m].X'}\ m \neq n$$

$$\frac{X \overset{a[m]}{\rightsquigarrow} X' \quad \mathsf{std}\,(Y)}{X + Y \overset{a[m]}{\rightsquigarrow} X' + Y} \qquad \frac{Y \overset{a[m]}{\rightsquigarrow} Y' \quad \mathsf{std}\,(X)}{X + Y \overset{a[m]}{\rightsquigarrow} X + Y'}$$

$$\frac{X \overset{a[m]}{\rightsquigarrow} X' \quad \mathsf{fsh}\,[m](Y)}{X \mid Y \overset{a[m]}{\rightsquigarrow} X' \mid Y} \qquad \frac{Y \overset{a[m]}{\rightsquigarrow} Y' \quad \mathsf{fsh}\,[m](X)}{X \mid Y \overset{a[m]}{\rightsquigarrow} X \mid Y'}$$

$$\frac{X \overset{a[m]}{\rightsquigarrow} X' \quad Y \overset{\overline{a}[m]}{\rightsquigarrow} Y'}{X \mid Y \overset{\tau[m]}{\rightsquigarrow} X' \mid Y'}\ (a \neq \tau)$$

$$\frac{X \overset{a[m]}{\rightsquigarrow} X'}{X \backslash A \overset{a[m]}{\rightsquigarrow} X' \backslash A}\ a \notin A \cup \overline{A} \qquad \frac{X \overset{a[m]}{\rightsquigarrow} X'}{X[f] \overset{f(a)[m]}{\rightsquigarrow} X'[f]}$$

Fig. 3. Reverse SOS rules for CCSK.

performed except as part of a communication. Suppose that $a.b$ communicates with $\overline{a}.d$ and then $a.c$ with $\overline{a}.e$. In CCSK we write this as follows:

$$P \overset{\tau[m]}{\rightarrow} (a[m].b \mid a.c \mid \overline{a}[m].d \mid \overline{a}.e)\backslash a \overset{\tau[n]}{\rightarrow} (a[m].b \mid a[n].c \mid \overline{a}[m].d \mid \overline{a}[n].e)\backslash a$$

Note that the process $a[m].b \mid a.c \mid \overline{a}[m].d \mid \overline{a}.e$ cannot regress by reversing $a[m]$ alone because key m is not fresh in $a.c \mid \overline{a}[m].d \mid \overline{a}.e$. The fact that m appears in $a.c \mid \overline{a}[m].d \mid \overline{a}.e$ which is in parallel with $a[m].b$ proves that the processes communicated a and $\overline{a}$.

Our notation does not allow us to backtrack by undoing a different pair of actions, but clearly we can change the order of reversing actions $\tau[m]$ and $\tau[n]$:

$$(a[m].b \mid a[n].c \mid \overline{a}[m].d \mid \overline{a}[n].e)\backslash a \overset{\tau[m]}{\rightsquigarrow} (a.b \mid a[n].c \mid \overline{a}.d \mid \overline{a}[n].e)\backslash a \overset{\tau[n]}{\rightsquigarrow} P$$

4.1. Related extensions of CCS

The present work is to be compared with Danos and Krivine's RCCS [11], but also in some sense to earlier approaches by Boudol and Castellani [6,7,9] and Degano et al. [14,15,16].

To aid comparison we give a simple example: the CCS processes $(a \mid \overline{a}.b)\backslash a$ and $\tau.b$. We might reasonably expect them to be equivalent, and indeed they are FR-bisimilar, as stated in Section 6. We have

$$(a \mid \overline{a}.b)\backslash a \stackrel{\tau[m]}{\rightarrow} (a[m] \mid \overline{a}[m].b)\backslash a \quad \text{and} \quad \tau.b \stackrel{\tau[m]}{\rightarrow} \tau[m].b.$$

In RCCS since $\langle\,\rangle \triangleright \nu a\ (a \mid \overline{a}.b) \equiv \nu a\ (\langle 1\rangle \triangleright a \mid \langle 2\rangle \triangleright \overline{a}.b)$ we write these transitions as

$$\nu a\ (\langle 1\rangle \triangleright a \mid \langle 2\rangle \triangleright \overline{a}.b) \stackrel{\langle 1\rangle,\langle 2\rangle:\tau}{\rightarrow} \nu a\ (\langle\langle 2\rangle, a, \mathbf{0}\rangle \cdot \langle 1\rangle \triangleright \mathbf{0} \mid \langle\langle 1\rangle, \overline{a}, \mathbf{0}\rangle \cdot \langle 2\rangle \triangleright b)$$

and

$$\langle\,\rangle \triangleright \tau.b \stackrel{\langle\rangle:\tau}{\rightarrow} \langle *, \tau, \mathbf{0}\rangle \triangleright b$$

respectively. In RCCS, transition labels contain extra information concerning which threads contribute. As a result it is harder to show that the processes are equivalent. Presumably one would have to abstract away from the thread information.

We might therefore say that, on the spectrum from intensionality to extensionality, the present work is more extensional than RCCS, though we shall see from the examples in Section 6 that CCSK definitely has a "true concurrency" flavour in terms of which processes it equates. We should however point out that RCCS has irreversible actions as well as reversible ones, and our remarks only apply to the irreversible actions. In their later work on transactions [12] Danos and Krivine define a notion of weak bisimulation where only irreversible actions (i.e. commit actions) are observable.

In [7] Boudol and Castellani compare three different non-interleaving models for CCS: proved transition systems (introduced earlier in [6]), event structures and Petri nets. To help this comparison, they propose *event transition systems*, where "events" are built using past actions, as an intermediate model. In event transition systems the whole structure of terms is preserved along execution, while their executed part is marked, so that the states ("marked terms") exactly represent a Petri net with a marking, or an event structure with a configuration. It was noted in [7] that "marked terms keep track of the dynamic operators as well as of the static ones". Similarly, the "ancestor" of a marked term corresponds to our notion of *root* (Definition 5.6). The main difference between our model and event transition systems seems to be the treatment of communication actions: marked terms do not record whether or not there has been a communication between complementary actions, whereas our terms with keys do. We illustrate below how terms of event transition systems keep track of the whole past of a transition by recording past actions and choices that have been made. These are recorded in the syntax of terms and, unlike in our approach, in the enhanced transition labels themselves. For example, where we write

$$(a \mid \overline{a}.b)\backslash a \stackrel{\tau[m]}{\rightarrow} (a[m] \mid \overline{a}[m].b)\backslash a,$$

in event transition systems this is

$$(a \mid \overline{a}.b)\backslash a \stackrel{\backslash a(a,\overline{a})}{\longrightarrow} (\underline{a} \mid \underline{\overline{a}}.b)\backslash a$$

and one needs to use additional rules to work out that the action label of the transition is τ. For sequential processes we write

$$a.b + c \stackrel{a[m]}{\rightarrow} a[m].b + c \stackrel{b[n]}{\rightarrow} a[m].b[n] + c$$

but in event transition systems both the enhanced labels and the extended syntax of CCS are used:

$$a.b + c \stackrel{+_0 a}{\rightarrow} \underline{a}.b +_0 c \stackrel{+_0 \underline{a}.b}{\rightarrow} \underline{a}.\underline{b} +_0 c$$

Note that the enhanced label $+_0\underline{a}.b$ represents syntactically in some sense the occurrence of b.

Additionally, Castellani employs a simplified version of event transition systems in [9] as the means to compare a static location transition system for CCS with a dynamic location transition system for CCS.

In [16], Degano and Priami present *enhanced transition systems*, which are strongly based on the proved transition systems discussed above and the earlier results in [14]. There, the labels of transitions are enriched with encodings of their deduction trees, and enhanced transition systems are used to analyse issues of distributivity and mobility of code.

Finally, a work on non-interleaving semantics for CCS by Degano et al. [15] is worth mentioning. They define a new transition relation that describes not only the actions that processes may perform in a given state but also the causal relation among subprocesses when the global state of the process changes. As a result, the arrow of the new transition relation is labelled by a pair: an action and a causality relation.

5. Properties of the transition relations

In this section we establish various properties of the forward and reverse transition relations defined earlier. In particular we show that the forward-reachable processes are closed under reverse transitions (Proposition 5.15); also that the new forward transition relation is in a sense conservative over the standard transition relation (Theorem 5.21).

We start by noting that the reverse transition relation inverts the forward transition relation:

Proposition 5.1. *Let* $P, P' \in \mathsf{Proc}$ *and* $\mu \in \mathsf{ActK}$. *Then* $P \stackrel{\mu}{\to} P'$ *iff* $P' \stackrel{\mu}{\rightsquigarrow} P$.

Proof. We show that if $P \stackrel{\mu}{\to} P'$ then $P' \stackrel{\mu}{\rightsquigarrow} P$ by induction on the proof of $P \stackrel{\mu}{\to} P'$. In fact if we take a proof of $P \stackrel{\mu}{\to} P'$ and just reverse each of the rules (replacing e.g. an instance of (1) by the corresponding instance of (1R)) then we get a proof of $P' \stackrel{\mu}{\rightsquigarrow} P$. This is because the reverse rules are just got by reversing the forward rules, and in particular predicates are only applied to inactive arguments, so that they hold equally before and after the transition.

The converse is similar. □

Each process has a set of keys. The set $\mathsf{keys}(P)$ of keys occurring in a process $P \in \mathsf{Proc}$ is defined as follows: $\mathsf{keys}(\mathbf{0}) \stackrel{\text{def}}{=} \emptyset$, $\mathsf{keys}(f(\overrightarrow{P})) \stackrel{\text{def}}{=} \bigcup_{i\in N} \mathsf{keys}(P_i)$ and $\mathsf{keys}(f_r[m](\overrightarrow{P})) \stackrel{\text{def}}{=} \{m\} \cup \bigcup_{i\in N} \mathsf{keys}(P_i)$. Clearly $P \in \mathsf{Std}$ iff $\mathsf{keys}(P) = \emptyset$. Also $\mathsf{fsh}[m](P)$ iff $m \notin \mathsf{keys}(P)$.

Any forward transition uses a fresh key:

Lemma 5.2. *Let* $P, P' \in \mathsf{Proc}$. *If* $P \stackrel{a[m]}{\to} P'$ *then* $m \notin \mathsf{keys}(P)$ *and* $\mathsf{keys}(P') = \mathsf{keys}(P) \cup \{m\}$.

Proof. By induction on the derivation of $P \stackrel{a[m]}{\to} P'$. □

Corollary 5.3. *Let* $P, P' \in \mathsf{Proc}$ *and* $s \in \mathsf{ActK}^*$. *If* $P \stackrel{s}{\to} P'$ *then* $\mathsf{keys}(P) \subseteq \mathsf{keys}(P')$, *no key is repeated in* s, *and* $\mathsf{keys}(s) = \mathsf{keys}(P') \setminus \mathsf{keys}(P)$ (*here* $\mathsf{keys}(s)$ *is defined in the obvious way*).

Proof. From Lemma 5.2. □

It follows immediately from Lemma 5.2 that standard processes are irreversible, and that the transition relation $\to$ is *well-founded*, meaning that there can be no infinite sequence of reverse transitions from any process. We call this the *Well-Foundedness (WF) Property*. Furthermore, in any purely forward (or purely reverse) computation, no label μ can be repeated, since no key can be repeated.

Let $P \to Q$ iff $P \stackrel{\mu}{\to} Q$ for some μ. Let $\to^*$ denote the reflexive and transitive closure of $\to$. Similarly for $\rightsquigarrow$.

Definition 5.4. A process $P \in \mathsf{Proc}$ is *reachable* if it can be reached by a finite sequence of forward transitions from a process in Std, i.e. there is $Q \in \mathsf{Std}$ such that $Q \to^* P$. Let Rch denote the set of reachable processes.

Of course, not every process is reachable. In CCSK, $a.b[m]$ is not reachable. A more interesting example is $a[m].b[n] \mid \overline{b}[n].\overline{a}[m]$. Here the names and keys match up, but there is a causal inconsistency.

It follows from Lemma 5.2 that if $P \in \mathsf{Rch}$ then every $\to$-computation from a process $Q \in \mathsf{Std}$ to P must have length $|\mathsf{keys}(P)|$.

Every subterm of a reachable process is reachable:

Proposition 5.5. *If* $P \in \mathsf{Rch}$ *and* P' *is a subterm of* P *then also* $P' \in \mathsf{Rch}$.

Proof. By induction on $|\mathsf{keys}(P)|$, with a sub-induction on the nesting depth of P as a term. If $|\mathsf{keys}(P)| = 0$ then $P \in \mathsf{Std}$. Hence $P' \in \mathsf{Std}$ and so $P' \in \mathsf{Rch}$.

So suppose that $|\mathsf{keys}(P)| = k+1$ and the result holds for k. There is $Q \in \mathsf{Rch}$ with $|\mathsf{keys}(Q)| = k$ such that $Q \stackrel{\mu}{\to} P$, for some $\mu \in \mathsf{ActK}$. By the induction hypothesis, every subterm of Q is reachable. Suppose that P is of the form $f(\overrightarrow{P})$. Then Q is of the form $f(\overrightarrow{Q})$. For each i, either $Q_i = P_i$ or $Q_i \stackrel{\mu_i}{\to} P_i$. If $Q_i = P_i$ then every subterm of P_i is reachable. Suppose that $Q_i \stackrel{\mu_i}{\to} P_i$. Since Q_i is reachable, so is P_i. If $|\mathsf{keys}(P_i)| \leq k$ then every subterm of P_i is reachable by the induction hypothesis. So suppose $|\mathsf{keys}(P_i)| = k+1$. Then every subterm of P_i is reachable by the sub-induction hypothesis, since P_i is of lower nesting depth than P. We conclude that every subterm of P is reachable.

If P is of the form $f_r[m](\overrightarrow{P})$ then the argument is similar, but more straightforward, since we always have $|\mathsf{keys}(P_i)| \leq k$. □

Since processes maintain their structure under $\to$, if a process P is reachable then we can determine by inspection from which standard process it must have been derived. We simply replace auxiliary operators $f_r[m]$ by the corresponding standard operators f. We call the resulting standard process the *root* of P.

Definition 5.6. For $P \in \mathsf{Proc}$ we define $\mathsf{rt}(P)$ (the *root* of P) to be the standard process got by structural induction as follows:

$$\mathsf{rt}(\mathbf{0}) \stackrel{\text{def}}{=} \mathbf{0} \qquad \mathsf{rt}(f(\overrightarrow{P})) \stackrel{\text{def}}{=} f(\overrightarrow{\mathsf{rt}(P)}) \qquad \mathsf{rt}(f_r[m](\overrightarrow{P})) \stackrel{\text{def}}{=} f(\overrightarrow{\mathsf{rt}(P)})$$

Clearly if $P \in \mathsf{Std}$ then $\mathsf{rt}(P) = P$. Computation preserves roots:

Lemma 5.7. *Let $P, Q \in \mathsf{Proc}$. If $P \stackrel{a[m]}{\to} Q$ then $\mathsf{rt}(P) = \mathsf{rt}(Q)$.*

Proof. By induction on the derivation of $P \stackrel{a[m]}{\to} Q$. □

Reachable processes can only have been derived from their root:

Lemma 5.8. *Let $P \in \mathsf{Proc}$ and $Q \in \mathsf{Std}$. If $Q \to^* P$ then $Q = \mathsf{rt}(P)$.*

Proof. By Lemma 5.7 we have $\mathsf{rt}(Q) = \mathsf{rt}(P)$. But $Q \in \mathsf{Std}$, and so $\mathsf{rt}(Q) = Q$. □

Two reachable processes which are connected by some sequence of forward or reverse transitions must have the same root:

Proposition 5.9. *Let $P, Q \in \mathsf{Rch}$. Then $P(\to \cup \rightsquigarrow)^* Q$ iff $\mathsf{rt}(P) = \mathsf{rt}(Q)$.*

Proof. ($\Rightarrow$) By Lemma 5.7 and Proposition 5.1.

($\Leftarrow$) Suppose that $\mathsf{rt}(P) = \mathsf{rt}(Q)$. By Lemma 5.8 there are computations $\mathsf{rt}(P) \to^* P$ and $\mathsf{rt}(Q) \to^* Q$. Hence $P \rightsquigarrow^* \mathsf{rt}(P) \to^* Q$ and so we obtain $P(\to \cup \rightsquigarrow)^* Q$. □

A "diamond" confluence property holds for reverse transitions:

Proposition 5.10 (Reverse Diamond (RD) Property). *Let P, Q and R belong to* Proc.

(1) *If $P \stackrel{a[m]}{\rightsquigarrow} Q$ and $P \stackrel{b[m]}{\rightsquigarrow} R$ then $a = b$ and $Q = R$.*

(2) *If $P \stackrel{a[m]}{\rightsquigarrow} Q$ and $P \stackrel{b[n]}{\rightsquigarrow} R$ with $m \neq n$, then there is S such that $Q \stackrel{b[n]}{\rightsquigarrow} S$ and $R \stackrel{a[m]}{\rightsquigarrow} S$.*

Proof. See Appendix A. □

Corollary 5.11. *Let P, Q and R belong to* Proc*, and let $s \in \mathsf{ActK}^*$, $\mu \in \mathsf{Act}$. If $P \stackrel{s}{\rightsquigarrow} Q$ and $p \stackrel{\mu}{\rightsquigarrow} R$ and $\mu \notin s$ then there is S such that $Q \stackrel{\mu}{\rightsquigarrow} S$ and $R \stackrel{s}{\rightsquigarrow} S$.*

Proof. By repeated use of Proposition 5.10(2). □

Proposition 5.10 implies that the reverse transition relation is finitely branching, since the number of reverse transitions of $P \in \mathsf{Proc}$ is bounded by $|\mathsf{keys}(P)|$.

We note that Proposition 5.10 implies: if $R \xrightarrow{\mu_1} \overset{\mu_2}{\rightsquigarrow} S$ then either $\mu_1 = \mu_2$ and $R = S$, or $R \overset{\mu_2}{\rightsquigarrow} \xrightarrow{\mu_1} S$. If we have a computation from P to Q involving both forward and reverse transitions, we can apply Proposition 5.10 repeatedly to any suitable adjacent transitions, to either cancel out or "promote" reverse transitions, producing a computation $P \overset{s}{\rightsquigarrow} \xrightarrow{t} Q$. Danos and Krivine call such computations *parabolic*. We have the following (cf. [11, Lemma 10]):

Lemma 5.12. *Let $P, Q \in \mathsf{Proc}$. Then $P(\rightarrow \cup \rightsquigarrow)^* Q$ iff $P \rightsquigarrow^* R \rightarrow^* Q$ for some R.*

Proof. (Sketch) It is enough to prove the forward implication, since the reverse implication is immediate. So suppose that $P(\rightarrow \cup \rightsquigarrow)^* Q$. Let us say that a reverse transition in a computation is *initial* if it is not preceded by any forward transition. We have to construct a new computation from P to Q in which all reverse transitions are initial. If in the starting computation there is no case of a forward transition being immediately followed by a reverse transition, then we are done.

So suppose that part of the computation is $R \xrightarrow{\mu_1} S \overset{\mu_2}{\rightsquigarrow} T$. If μ_1 and μ_2 have the same key then by Proposition 5.10(1) we have $\mu_1 = \mu_2$ and $R = T$. So we can eliminate the $R \xrightarrow{\mu_1} S$ and $S \overset{\mu_2}{\rightsquigarrow} T$ transitions to get a new shorter computation from P to Q. If μ_1 and μ_2 have different keys then by Proposition 5.10(2) there is U such that $R \overset{\mu_2}{\rightsquigarrow} U \xrightarrow{\mu_1} T$. We replace $R \xrightarrow{\mu_1} S \overset{\mu_2}{\rightsquigarrow} T$ by $R \overset{\mu_2}{\rightsquigarrow} U \xrightarrow{\mu_1} T$ to get a new computation from P to Q.

By repeatedly performing the above procedure, we either cancel out non-initial reverse transitions or else we move them nearer to the start of the computation, until they eventually become initial. Eventually there will be no more cases of a forward transition being immediately followed by a reverse transition, and we then have constructed the required parabolic computation. □

The Reverse Diamond Property leads to a corresponding result for sequences of reverse transitions. First a definition:

Definition 5.13. Let $s, t \in \mathsf{ActK}^*$. Then $s\backslash t$ is s with all $\mu \in t$ removed. More formally

$$\varepsilon\backslash t \overset{\text{def}}{=} \varepsilon \qquad (\mu s)\backslash t \overset{\text{def}}{=} \begin{cases} s\backslash t & \text{if } \mu \in t \\ \mu(s\backslash t) & \text{if } \mu \notin t \end{cases}$$

Lemma 5.14. *If $P \overset{s}{\rightsquigarrow} Q$ and $P \overset{t}{\rightsquigarrow} R$ then there is S such that $Q \overset{t\backslash s}{\rightsquigarrow} S$ and $R \overset{s\backslash t}{\rightsquigarrow} S$.*

Proof. By induction on s. If $s = \varepsilon$ then $P = Q$ and we let $S = R$. Suppose $s = \mu s'$. We have $P \overset{\mu}{\rightsquigarrow} P' \overset{s'}{\rightsquigarrow} Q$ for some P'. There are two cases:

(1) $\mu \in t$. Suppose $t = t_1 \mu t_2$. By Corollary 5.11 and Proposition 5.10(1), we get $P' \overset{t_1 t_2}{\rightsquigarrow} R$. By induction there is S such that $Q \overset{(t_1 t_2)\backslash s'}{\rightsquigarrow} S$ and $R \overset{s'\backslash(t_1 t_2)}{\rightsquigarrow} S$. But $(t_1 t_2)\backslash s' = t\backslash s$ and $s'\backslash(t_1 t_2) = s\backslash t$.

(2) $\mu \notin t$. By Corollary 5.11, we get R' such that $R \overset{\mu}{\rightsquigarrow} R'$ and $P' \overset{t}{\rightsquigarrow} R'$. By induction there is S such that $Q \overset{t\backslash s'}{\rightsquigarrow} S$ and $R' \overset{s'\backslash t}{\rightsquigarrow} S$. But $t\backslash s' = t\backslash s$. Also $R \overset{\mu(s'\backslash t)}{\rightsquigarrow} S$, and $\mu(s'\backslash t) = s\backslash t$. Hence S is as required. □

The reachable terms are closed under reverse transitions, meaning that a process can never reverse into an "inconsistent" past as shown in the next proposition. The inclusion of predicates in simple SOS rules may lead to a failure of this property; see operator h in Section 7.1.

Proposition 5.15. *If $P \in \mathsf{Rch}$, $\mu \in \mathsf{ActK}$ and $P \overset{\mu}{\rightsquigarrow} P'$ then $P' \in \mathsf{Rch}$.*

Proof. If $P \in \mathsf{Rch}$ then there is a computation $\mathsf{rt}(P) \xrightarrow{s} P$. By Lemma 5.14 there is Q such that $Q \xrightarrow{s\backslash\mu} P'$ and $Q \xrightarrow{\mu\backslash s} \mathsf{rt}(P)$. But then we must have $Q = \mathsf{rt}(P)$, since $\mathsf{rt}(P) \in \mathsf{Std}$. Hence $P' \in \mathsf{Rch}$ as required. □

Starting at a process P, any two reverse computations which are permutations of each other will yield the same resultant process, as the next result states:

Proposition 5.16. *Let $P, Q, R \in \mathsf{Proc}$ and $s, t \in \mathsf{ActK}^*$. If $P \overset{s}{\rightsquigarrow} Q$ and $P \overset{t}{\rightsquigarrow} R$ and t is a permutation of s, then $Q = R$.*

Proof. By Lemma 5.14. □

We next show that any two forward computations with the same start and endpoint are *homotopic*, meaning that one computation can be transformed into the other by local swaps.

Definition 5.17 ([19, *Definition* 3.1]). Two (finite) forward computations in a ltr are *adjacent* if one can be got from the other by replacing some segment $P \overset{\mu}{\rightarrow} R \overset{\nu}{\rightarrow} Q$ by $P \overset{\nu}{\rightarrow} S \overset{\mu}{\rightarrow} Q$. Two forward computations are *homotopic* if they are related by the reflexive and transitive closure of adjacency.

Proposition 5.18. *Let $P, Q \in \mathsf{Proc}$ and suppose that $P \overset{s}{\rightarrow} Q$, $P \overset{t}{\rightarrow} Q$. Then the two computations are homotopic.*

Proof. By Corollary 5.3, $\mathsf{keys}(s) = \mathsf{keys}(t) = \mathsf{keys}(Q) \setminus \mathsf{keys}(P)$. Let $k = |\mathsf{keys}(Q) \setminus \mathsf{keys}(P)|$. We proceed by induction on k.

For $k = 0$ there is nothing to show. Suppose that $s = s'.a[m]$ and $t = t'.b[n]$. There are R, R' such that $P \overset{s'}{\rightarrow} R \overset{a[m]}{\rightarrow} Q$ and $P \overset{t'}{\rightarrow} R' \overset{b[n]}{\rightarrow} Q$.

If $m = n$ then by Proposition 5.10 we have $a = b$ and $R = R'$. By the induction hypothesis $P \overset{t'}{\rightarrow} R'$ is homotopic to $P \overset{s'}{\rightarrow} R$, and so $P \overset{t}{\rightarrow} Q$ is homotopic to $P \overset{s}{\rightarrow} Q$ as required.

So suppose $m \neq n$. By Proposition 5.10 there is S such that $R \overset{b[n]}{\rightsquigarrow} S$ and $R' \overset{a[m]}{\rightsquigarrow} S$. We know that n occurs as a key in s'. Suppose $s' = s_1.c[n].s_2$ for some s_1, s_2, c. Note that n does not occur in s_2. There are T, U such that $P \overset{s_1}{\rightarrow} T \overset{c[n]}{\rightarrow} U \overset{s_2}{\rightarrow} R$. By Lemma 5.14 we get T' such that $T' \overset{b[n]}{\rightarrow} U$ and $T' \overset{s_2}{\rightarrow} S$. But then again by Proposition 5.10 we have $b = c$ and $T' = T$. Hence $P \overset{s_1}{\rightarrow} T \overset{s_2}{\rightarrow} S$.

By the induction hypothesis $P \overset{s'}{\rightarrow} R \overset{a[m]}{\rightarrow} Q$ is homotopic to $P \overset{s_1 s_2}{\rightarrow} S \overset{b[n]}{\rightarrow} R \overset{a[m]}{\rightarrow} Q$, which is adjacent to $P \overset{s_1 s_2}{\rightarrow} S \overset{a[m]}{\rightarrow} R' \overset{b[n]}{\rightarrow} Q$, which is homotopic to $P \overset{t'}{\rightarrow} R' \overset{b[n]}{\rightarrow} Q$, again by the induction hypothesis. Hence the result. □

A special case of Proposition 5.18 is when s and t are of length one: if $P \overset{\mu}{\rightarrow} Q$ and $P \overset{\nu}{\rightarrow} Q$ then $\mu = \nu$. We call this the *Unique Transition (UT) Property*.

The analogue of Proposition 5.10 does not hold for forward transitions, since two forward transitions $P \overset{a[m]}{\rightarrow} Q$ and $P \overset{b[n]}{\rightarrow} R$ may conflict. However we can complete the diamond if the forward transitions are joinable, in the sense that Q and R can reach a common process S by forward moves:

Proposition 5.19 (Forward Diamond (FD) Property). *Let P, Q, R and T be members of* Proc.

(1) *If $P \overset{a[m]}{\rightarrow} Q \overset{s}{\rightarrow} T$ and $P \overset{b[m]}{\rightarrow} R \overset{t}{\rightarrow} T$ then $a = b$ and $Q = R$.*

(2) *If $P \overset{a[m]}{\rightarrow} Q \overset{s}{\rightarrow} T$ and $P \overset{b[n]}{\rightarrow} R \overset{t}{\rightarrow} T$ with $m \neq n$, then there is S such that $Q \overset{b[n]}{\rightarrow} S$, $R \overset{a[m]}{\rightarrow} S$ and $S \overset{s \setminus b[n]}{\rightarrow} T$, $S \overset{t \setminus a[m]}{\rightarrow} T$.*

Proof. See Appendix B. □

In Proposition 5.19(1) we do need Q and R to be joinable. Consider, for instance, the CCS process $a + b$ (where $a \neq b$). We have $a + b \overset{a[m]}{\rightarrow} a[m] + b$ and $a + b \overset{b[m]}{\rightarrow} a + b[m]$.

We now turn to showing that the new forward transition relation $\rightarrow$ is essentially conservative over the standard transition relation $\rightarrow_{\mathsf{S}}$. We have to take into account the fact that we have introduced auxiliary operators and keys.

A nonstandard process can be converted to a corresponding standard process by "pruning" the auxiliary operators (cf. the forgetful map of [11]):

Definition 5.20. The pruning map $\pi : \mathsf{Proc} \rightarrow \mathsf{Std}$ is defined as follows:

$$\pi(\mathbf{0}) \stackrel{\text{def}}{=} \mathbf{0}$$

$$\pi(f(\overrightarrow{P})) \stackrel{\text{def}}{=} \begin{cases} \pi(P_d) & \text{if } d \in D_f \wedge \neg\mathsf{std}(P_d) \wedge \forall e \in E_f \setminus \{d\}.\, \mathsf{std}(P_e) \\ f(\overrightarrow{\pi(P)}) & \text{if } \forall e \in E_f.\, \mathsf{std}(P_e) \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$\pi(f_r[m](\overrightarrow{P})) \stackrel{\text{def}}{=} \begin{cases} \pi(P_{\mathsf{ta}(r)}) & \text{if } \forall e \in E_f \setminus \{\mathsf{ta}(r)\}.\, \mathsf{std}(P_e) \\ \mathbf{0} & \text{otherwise} \end{cases}$$

for any choice axiom r for f, and where $\overrightarrow{\pi(P)}$ is the vector $\pi(P_1), \ldots, \pi(P_n)$.

Clearly, if $P \in \mathsf{Std}$ then $\pi(P) = P$. It can easily be shown that the third case for $\pi(f(\overrightarrow{P}))$ and the second case for $\pi(f_r(\overrightarrow{P}))$ will not arise with reachable terms.

Theorem 5.21 (Conservation). *Suppose* $P \in \mathsf{Proc}$.

(1) *If* $P \stackrel{a[m]}{\rightarrow} P'$ *then* $\pi(P) \stackrel{a}{\rightarrow}_{\mathsf{S}} \pi(P')$.

(2) *If* $\pi(P) \stackrel{a}{\rightarrow}_{\mathsf{S}} P'$ *then for any* $m \in \mathcal{K} \setminus \mathsf{keys}(P)$ *there is* P'' *such that* $P \stackrel{a[m]}{\rightarrow} P''$ *and* $\pi(P'') = P'$.

Proof. See Appendix C. □

6. Forward–reverse bisimulation

We can show that the reversible transition relation $\rightarrow$ induces essentially the same bisimulation equivalence on processes as the standard transition relation $\rightarrow_{\mathsf{S}}$. We first recall standard strong bisimulation on the standard terms:

Definition 6.1. A symmetric relation $\mathcal{S}$ on Std is an *S-bisimulation* if whenever $\mathcal{S}(P, Q)$ then if $P \stackrel{a}{\rightarrow}_{\mathsf{S}} P'$ then there is Q' such that $Q \stackrel{a}{\rightarrow}_{\mathsf{S}} Q'$ and $\mathcal{S}(P', Q')$. We define $P \sim_{\mathsf{S}} Q$ iff there is an S-bisimulation $\mathcal{S}$ such that $\mathcal{S}(P, Q)$.

The corresponding notion for forward transitions on Proc and predicates Pred is

Definition 6.2. A symmetric relation $\mathcal{S}$ on Proc is an *F-bisimulation* if $\mathcal{S}(P, Q)$ implies

- $\mathsf{p}(P) \Leftrightarrow \mathsf{p}(Q)$ for all $\mathsf{p} \in \mathsf{Pred}$;
- if $P \stackrel{\mu}{\rightarrow} P'$ then there is Q' such that $Q \stackrel{\mu}{\rightarrow} Q'$ and $\mathcal{S}(P', Q')$.

We define $P \sim_{\mathsf{F}} Q$ iff there is an F-bisimulation $\mathcal{S}$ such that $\mathcal{S}(P, Q)$.

Note that the first item in Definition 6.2 could be written as $\mathsf{keys}(P) = \mathsf{keys}(Q)$, since $\mathsf{fsh}[m](P) \Leftrightarrow m \notin \mathsf{keys}(P)$ and $\mathsf{std}(P) \Leftrightarrow \mathsf{keys}(P) = \emptyset$.

F-bisimulation is conservative over S-bisimulation by the following result:

Proposition 6.3. *Let* $P, Q \in \mathsf{Proc}$. *Then the following are equivalent*:

(1) $P \sim_{\mathsf{F}} Q$

(2) $\pi(P) \sim_{\mathsf{S}} \pi(Q)$ *and* $\mathsf{p}(P) \Leftrightarrow \mathsf{p}(Q)$ *for all* $\mathsf{p} \in \mathsf{Pred}$

Proof. From Theorem 5.21. □

An immediate consequence of Proposition 6.3 is that $\sim_{\mathsf{S}}$ and $\sim_{\mathsf{F}}$ coincide on standard processes.

Since all rules for operators in Proc are in the *path* format, the congruence result in [3] implies the following result:

Proposition 6.4. *The relation $\sim_{\mathsf{F}}$ is a congruence with respect to all the operators of* Proc.

We now define bisimulation for both forward and reverse transitions:

Definition 6.5. A symmetric relation $\mathcal{S}$ on Proc is a *forward–reverse (FR) bisimulation* if whenever $\mathcal{S}(P, Q)$ then
- $\mathsf{p}(P) \Leftrightarrow \mathsf{p}(Q)$ for all $\mathsf{p} \in \mathsf{Pred}$;
- if $P \xrightarrow{\mu} P'$ then there is Q' such that $Q \xrightarrow{\mu} Q'$ and $\mathcal{S}(P', Q')$;
- if $P \overset{\mu}{\rightsquigarrow} P'$ then there is Q' such that $Q \overset{\mu}{\rightsquigarrow} Q'$ and $\mathcal{S}(P', Q')$.

We define $P \sim_{\mathsf{FR}} Q$ iff there is an FR bisimulation $\mathcal{S}$ such that $\mathcal{S}(P, Q)$.

Proposition 6.6. *Let $P, Q \in$ Proc. If $P \sim_{\mathsf{FR}} Q$ then $P \sim_{\mathsf{F}} Q$.*

The converse does not hold. For instance in CCSK we have $a \mid a \sim_{\mathsf{F}} a.a$, but $a \mid a \not\sim_{\mathsf{FR}} a.a$. This is because $a \mid a \xrightarrow{a[m]} \xrightarrow{a[n]} a[m] \mid a[n] \overset{a[m]}{\rightsquigarrow} a \mid a[n]$ and $m \neq n$. This sequence of transitions cannot be matched by $a.a$: we have

$$a.a \xrightarrow{a[m]} \xrightarrow{a[n]} a[m].a[n] \overset{a[m]}{\not\rightsquigarrow}.$$

Similarly $a \mid b \sim_{\mathsf{F}} a.b + b.a$, but $a \mid b \not\sim_{\mathsf{FR}} a.b + b.a$.

As an example of processes which are equivalent under FR bisimulation, note that for any $P \in$ Std, $P + P \sim_{\mathsf{FR}} P$. We can also show that for any $P \in$ Std, $(a \mid \overline{a}.P)\backslash a \sim_{\mathsf{FR}} \tau.(P\backslash a)$.

Finally, we show that forward–reverse bisimulation is preserved by operators of arbitrary reversible process calculi with keys:

Theorem 6.7. *The relation $\sim_{\mathsf{FR}}$ is a congruence with respect to all the operators of* Proc.

Proof. The proof method is standard as, for example, in [18,3]. It relies on checking the properties of forward–reverse bisimulation and this is straightforward due to a simple form of forward and reverse rules. The details are given in Appendix D. □

Several notions of bisimulation taking into account backward as well as forward moves have been discussed in the literature. The *back and forth bisimulation* of [13] is constrained to only go back along the path that brought a process to its current state. The authors (acknowledging a discussion with Colin Stirling) observe that if a different reverse path could be followed then $a \mid b$ could be distinguished from $a.b + b.a$. Back and forth bisimulation where any reverse path can be followed is discussed in [5] both for transition systems and event structures. Essentially the same notion, but called *backward-forward bisimulation*, is defined in [17] for occurrence transition systems. The non-interleaving semantics community has proposed several bisimulation-like equivalences [20] and we intend to investigate how FR bisimulation compares with them.

7. Possible extensions

Our conversion procedure applies only to operators that can be defined by SOS rules in the simple path format (Definition 2.1). However, there are process operators with SOS rules which go beyond the mentioned format. For example, sequential composition of ACP [4] uses predicates in SOS rules and recursion is conveniently dealt with by adding structural congruence to rules. This section describes only preliminary ideas on how the conversion procedure can be extended so that it applies to a wider class of operators. We provide no technical results here; we intend to address the problem of extensions more fully in a follow-up paper.

ACP action constants can be defined analogously to prefixing of CCS. We have the constant ε (successful termination) and constants a for each $a \in \mathsf{Act}$. The defining rules $a \xrightarrow{a}_{\mathrm{S}} \varepsilon$ are converted to $a \xrightarrow{a[m]} a[m]$, where $a[m]$ are auxiliary constants for all $m \in \mathcal{K}$. There are no forward SOS rules for the auxiliary constants and no transition rules for ε.

7.1. Predicates

The next extension would be to allow predicates in SOS rules. An example is the successful termination predicate trm in the rules for ACP's sequential composition "$\cdot$" below [4]. Care needs to be taken when adding predicates to premises in order to avoid *lookahead* in the reverse rules. For the majority of operators with predicates the converted and reverse rules are very natural.

$$\frac{X \xrightarrow{a}_{\mathrm{S}} X'}{X \cdot Y \xrightarrow{a}_{\mathrm{S}} X' \cdot Y} \qquad \frac{Y \xrightarrow{b}_{\mathrm{S}} Y' \quad \mathsf{trm}(X)}{X \cdot Y \xrightarrow{b}_{\mathrm{S}} Y'}$$

With some simplifications, the converted and reverse rules are

$$\frac{X \xrightarrow{a} X' \quad \mathsf{std}(Y)}{X \cdot Y \xrightarrow{a} X' \cdot Y} \qquad \frac{Y \xrightarrow{b} Y' \quad \mathsf{trm}(X)}{X \cdot Y \xrightarrow{b} X \cdot Y'}$$

$$\frac{X \overset{a}{\rightsquigarrow} X' \quad \mathsf{std}(Y)}{X \cdot Y \overset{a}{\rightsquigarrow} X' \cdot Y} \qquad \frac{Y \overset{b}{\rightsquigarrow} Y' \quad \mathsf{trm}(X)}{X \cdot Y \overset{b}{\rightsquigarrow} X \cdot Y'}$$

Here we extend trm to cover nonstandard processes.

In general, the inclusion of predicates in standard rules may lead to a loss of some of the properties introduced in Section 5. We illustrate this with several examples.

Consider the Unique Transition property (defined immediately after Proposition 5.18). If static rules are extended with arbitrary predicates then UT no longer holds. Assume that $\mathsf{p}(a)$ and $\mathsf{q}(a)$ hold and operator f has the following rules:

$$\frac{X \xrightarrow{a}_{\mathrm{S}} X' \quad \mathsf{p}(Y)}{f(X,Y) \xrightarrow{a}_{\mathrm{S}} f(X',Y)} \qquad \frac{X \xrightarrow{a}_{\mathrm{S}} X' \quad \mathsf{q}(Y)}{f(X,Y) \xrightarrow{b}_{\mathrm{S}} f(X',Y)}$$

Consider $f(a,a)$. By the converted first rule $f(a,a) \xrightarrow{a[m]} f(a[m],a)$, and by the converted second rule $f(a,a) \xrightarrow{b[m]} f(a[m],a)$, thus failing UT. To avoid this problem we could demand the following (strong) condition: If two rules have transition premises for the same arguments, then they must have the same predicate premises and the same conclusion. This condition alone is not sufficient since we can find simple operators that satisfy it but fail, for example, the Reverse Diamond property.

Consider the following operator h. The second rule contains the successful termination predicate trm defined above. The rules for h are

$$\frac{X \xrightarrow{a}_{\mathrm{S}} X'}{h(X,Y) \xrightarrow{a}_{\mathrm{S}} h(X',Y)} \qquad \frac{Y \xrightarrow{b}_{\mathrm{S}} Y' \quad \mathsf{trm}(X)}{h(X,Y) \xrightarrow{b}_{\mathrm{S}} h(X,Y')}$$

Clearly, these rules are static and the second rule schema satisfies the condition proposed to prevent the failure of UT. After we convert the rules, we deduce $h(a,b) \xrightarrow{a[m]} \xrightarrow{b[n]} h(a[m],b[n])$ and $h(a[m],b[n]) \overset{a[m]}{\rightsquigarrow} h(a,b[n])$. But it is not the case that $h(a,b[n]) \overset{b[n]}{\rightsquigarrow}$ since $\mathsf{trm}(a)$ is not valid. As a result RD fails. Also, $h(a,b[n])$ is not reachable from $h(a,b)$ by forward actions only. Hence, Proposition 5.15 is not valid for simple calculi with operators like h.

Similarly, we can find examples of standard simple operators with rules that use the successful termination predicate trm in the premises that fail the Forward Diamond property. This, and the examples above, shows that further investigation is needed to find a form of predicates that can be used safely in the premises of rules for simple process operators.

Finally, to allow the external choice operator of CSP, "$\Box$", we need to relax the condition that static arguments cannot be dynamic. The defining rules for "$\Box$" are given below, where the last two rules are rule schemas for all $a \in \mathsf{Act} \setminus \{\tau\}$

$$\frac{X \xrightarrow{\tau}_{\mathrm{S}} X'}{X \,\square\, Y \xrightarrow{\tau}_{\mathrm{S}} X' \,\square\, Y} \qquad \frac{Y \xrightarrow{\tau}_{\mathrm{S}} Y'}{X \,\square\, Y \xrightarrow{\tau}_{\mathrm{S}} X \,\square\, Y'}$$

$$\frac{X \xrightarrow{a}_{\mathrm{S}} X'}{X \,\square\, Y \xrightarrow{a}_{\mathrm{S}} X'} \qquad \frac{Y \xrightarrow{a}_{\mathrm{S}} Y'}{X \,\square\, Y \xrightarrow{a}_{\mathrm{S}} Y'}$$

By introducing an auxiliary predicate $\mathsf{before}(P)$, which holds if $P \in \mathsf{Std}$ or P is a derivative from a standard term via a sequence of silent actions, we obtain the following converted and reverse rules:

$$\frac{\mathsf{before}(Y) \quad X \xrightarrow{\mu} X'}{X \,\square\, Y \xrightarrow{\mu} X' \,\square\, Y} \qquad \frac{\mathsf{before}(X) \quad Y \xrightarrow{\mu} Y'}{X \,\square\, Y \xrightarrow{\mu} X \,\square\, Y'}$$

$$\frac{\mathsf{before}(Y) \quad X \overset{\mu}{\rightsquigarrow} X'}{X \,\square\, Y \overset{\mu}{\rightsquigarrow} X' \,\square\, Y} \qquad \frac{\mathsf{before}(X) \quad Y \overset{\mu}{\rightsquigarrow} Y'}{X \,\square\, Y \overset{\mu}{\rightsquigarrow} X \,\square\, Y'}$$

7.2. Recursion

Simple process languages can be extended with recursion by adding process constants A, defined by equations $A \overset{\mathrm{def}}{=} P$, and following a general approach for adding structural congruence $\equiv$ to SOS rules due to Mousavi and Reniers [26]. This requires an adaptation of the standard notions of SOS rules, proof and provable transitions as in [3]. We describe this adaptation briefly below, and give only the basic idea on how to reverse simple process languages with recursion.

Let $L_{\mathrm{S}} = (\Sigma_{\mathrm{S}}, \mathsf{Act}, R_{\mathrm{S}})$ be a simple process language extended with the set $\mathcal{C}$ of process constants, ranged over by A and B, such that each constant A is accompanied by a definition $A \overset{\mathrm{def}}{=} P$, where P is a closed term over the extended language. We define a structural congruence $\equiv_{\mathrm{S}}$ over the terms of the extended language as the smallest equivalence and congruence relation (with respect to all operators) generated by the laws

$$A \equiv_{\mathrm{S}} P$$

for all constants A in $\mathcal{C}$. Note that these structural congruence equations adhere to the form of the *defining equations* belonging to the cfsc format [26], a format which is safe for bisimulation. In order to incorporate structural congruence in SOS rules, a special rule is added [25,26], called the *Structural Congruence Rule*, where $a \in \mathsf{Act}$:

$$\frac{X \equiv_{\mathrm{S}} Y \quad Y \xrightarrow{a}_{\mathrm{S}} Y' \quad Y' \equiv_{\mathrm{S}} X'}{X \xrightarrow{a}_{\mathrm{S}} X'} \ \mathsf{struct}$$

This rule, however, does not fit into the form of SOS rules in Definition 2.1 and, hence, we cannot employ the standard method for associating an ltr to L_{S}. To overcome this problem, we follow closely the approach in [26]; the form of permitted SOS rules is extended to include expressions $t_j \equiv_{\mathrm{S}} t'_j$, where t_j and t'_j are arbitrary terms in the language. Moreover, the notions of proof and provable transitions as, for example, in [3] are extended to account for the presence of $t_j \equiv_{\mathrm{S}} t'_j$ expressions in SOS rules.

Let R_{struct} be the set of the struct rules for all $a \in \mathsf{Act}$. Thus, we have obtained a *simple process language with recursion* $L_{\mathrm{Srec}} = (\Sigma_{\mathrm{S}} \cup \mathcal{C}, \mathsf{Act}, R_{\mathrm{S}} \cup R_{\mathsf{struct}})$. We associate an ltr with this language following the standard method extended as above. By a general congruence result in [26] we deduce that S-bisimulation (see Section 6) is a congruence for process languages with recursion like L_{Srec}.

The conversion procedure from Definition 3.7 is then extended to languages such as L_{Srec} in the following way to give a reversible process calculus with keys and recursion L_{rec}. Firstly, a new structural congruence $\equiv$ is defined on L_{rec} terms as the smallest equivalence and congruence relation (with respect to operators of L_{rec}) generated by laws

$$A \equiv_{\mathrm{S}} P$$

for all constants A in $\mathcal{C}$, where P are closed terms over L_{Srec} (which do not contain any auxiliary operators). The signature of L_{rec} is $\Sigma_{\mathrm{S}} \cup \mathcal{C} \cup \Sigma_{\mathrm{A}}$, with Σ_{A} as in Definition 3.7. The set of forward rules of L_{rec} consists of R_{F}, as in Definition 3.7, and R_{structk} which is the set of rules below for all $a[m] \in \mathsf{ActK}$.

$$\frac{X \equiv Y \quad Y \stackrel{a[m]}{\rightarrow} Y' \quad Y' \equiv X'}{X \stackrel{a[m]}{\rightarrow} X'} \ \mathsf{structk}$$

Here, the subset R_{P} of R_{F} contains also the following rules for constants A:

$$\frac{}{\mathsf{std}(A)} \qquad \frac{}{\mathsf{fsh}[m](A)}$$

The set of action labels of L_{rec} is ActK. Finally, the set of reverse rules of L_{rec} consists of R_{R}, as in Definition 3.7, and R_{structkR}, the set of rules structkR below, which are the symmetric versions of the rules in R_{structk}:

$$\frac{X \equiv Y \quad Y \stackrel{a[m]}{\rightsquigarrow} Y' \quad Y' \equiv X'}{X \stackrel{a[m]}{\rightsquigarrow} X'} \ \mathsf{structkR}$$

Again, following the standard method extended as above to take into account the presence of $\equiv$, we associate the forward and the reverse ltrs with L_{rec}. We conjecture that simple process calculi with recursion like L_{rec} enjoy all relevant properties established in previous sections, with $=$ replaced by $\equiv$.

As an example of computation involving recursion consider constant A defined by $A \stackrel{\mathrm{def}}{=} a.A + b$. Since $A \equiv a.A + b$ and $a.A + b \stackrel{a[m]}{\rightarrow} a[m].A + b$ and $a[m].A + b$ is congruent to itself, we obtain by structk

$$A \stackrel{a[m]}{\rightarrow} a[m].A + b.$$

Next, we can unwind recursion once more and, by combining structk with the rule for $a[m].X$ (which says that $\equiv$ is a congruence), we obtain

$$a[m].(a.A + b) + b \stackrel{b[n]}{\rightarrow} a[m].(a.A + b[n]) + b.$$

8. Conclusions

There has been much recent interest in reversible computing, including the pioneering work of Danos and Krivine on reversible CCS. We have introduced a method for converting standard irreversible operators of algebraic process calculi such as CCS into reversible operators. As far as we are aware, this is the first time that such a method has been proposed in the context of general process calculi. Our method works on operators with rules of a simple form. We arrive at new rules which preserve the structure of the terms. An important feature of our method is the introduction of keys to bind synchronised actions together. We have also obtained an appropriate notion of bisimulation on terms and we have proved that it is preserved by all reversible operators. Our work demonstrates that it is possible to make many standard operators reversible in a manner which is both algebraic and tractable.

Future work includes adding in irreversible transitions (already included in RCCS [11,12]) and extending our format with predicates. We also aim to take a more abstract approach and formulate a definition of reversible transition relations in general (the present work being a concrete example). Finally, we intend to investigate the links between reversibility and true concurrency.

Acknowledgments

We wish to thank Philippa Gardner, Daniele Varacca, Nobuko Yoshida, Shoji Yuen and the FOSSACS 2006 and JLAP referees for helpful discussions, comments and suggestions. The second author would like to thank the University of Leicester for granting study leave, and acknowledge gratefully support from EPSRC, grant EP/C001885/1, and from Nagoya University during a research visit. Fig. 1 was drawn using Paul Taylor's commutative diagram macro package.

Appendix

A. Proof of Proposition 5.10

We use structural induction on P and prove both parts at once.

(1) If $P = \mathbf{0}$ then no reverse transitions are possible and we are done.

(2) Suppose $P = f(\overrightarrow{P})$. Plainly reverse transitions from P can only come from rules of type (1R) or (2R).

(a) Suppose that $P \overset{a[m]}{\rightsquigarrow} Q$ is derived from $\{P_i \overset{a_i[m]}{\rightsquigarrow} Q_i\}_{i\in I}$ via a rule of type (1R):

$$(*) \qquad \frac{\{X_i \overset{a_i[m]}{\rightsquigarrow} X'_i\}_{i\in I} \quad \{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S\setminus I}}{f(\overrightarrow{X}) \overset{a[m]}{\rightsquigarrow} f(\overrightarrow{X'})}$$

Since $\mathsf{std}(P_e)$ for all $e \in E$, $P \overset{b[n]}{\rightsquigarrow} R$ must be derived from a rule of type (1R) rather than (2R). So suppose that $P \overset{b[n]}{\rightsquigarrow} R$ is derived from $\{P_i \overset{b_i[n]}{\rightsquigarrow} R_i\}_{i\in I'}$ via:

$$(**) \qquad \frac{\{X_i \overset{b_i[n]}{\rightsquigarrow} X'_i\}_{i\in I'} \quad \{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[n](X_i)\}_{i\in S\setminus I'}}{f(\overrightarrow{X}) \overset{b[n]}{\rightsquigarrow} f(\overrightarrow{X'})}$$

Suppose first that $m = n$. From the freshness conditions it is easy to deduce that $I = I'$. By induction, $a_i = b_i$ and $Q_i = R_i$ for each $i \in I$. By the condition that two rules of type (1R) with the same premises must have the same conclusion, we deduce that $a = b$. Also $Q = f(\overrightarrow{Q}) = f(\overrightarrow{R}) = R$ as required.

Now suppose that $m \neq n$. By induction, for any $i \in I \cap I'$ there is S_i such that $Q_i \overset{b_i[n]}{\rightsquigarrow} S_i$ and $R_i \overset{a_i[m]}{\rightsquigarrow} S_i$. For $i \in I \setminus I'$ let $S_i = Q_i$. For $i \in I' \setminus I$ let $S_i = R_i$. Finally for $i \in N \setminus (I \cup I')$ let $S_i = P_i$. Let $S = f(\overrightarrow{S})$. We claim that $Q = f(\overrightarrow{Q}) \overset{b[n]}{\rightsquigarrow} f(\overrightarrow{S}) = S$ via rule (**). It is easy to check that $Q_i \overset{b_i[n]}{\rightsquigarrow} R_i$ for all $i \in I'$ (there are two cases, depending on whether or not $i \in I$). Also clearly $Q_e = P_e$ for each $e \in E$, so that $\mathsf{std}(Q_e)$ holds for all $e \in E$. It remains to check that $\mathsf{fsh}[n](Q_i)$ holds for each $i \in S \setminus I'$. We know $\mathsf{fsh}[n](P_i)$ for each $i \in S \setminus I'$. If $i \in I$ then $P_i \overset{a_i[m]}{\rightsquigarrow} Q_i$, and so $\mathsf{fsh}[n](Q_i)$ by Lemma 5.2. If $i \notin I$ then $Q_i = P_i$ and so again $\mathsf{fsh}[n](Q_i)$. Hence rule (**) applies to $f(\overrightarrow{Q})$ and we have $Q \overset{b[n]}{\rightsquigarrow} S$ as required.

Similarly $R \overset{a[m]}{\rightsquigarrow} S$ via rule (*).

(b) Suppose that $P \overset{a[m]}{\rightsquigarrow} Q$ is derived from $P_d \overset{a[m]}{\rightsquigarrow} Q_d$ via rule (2R). Then we must have $\mathsf{std}(P_e)$ for all $e \in E$ except d, and not $\mathsf{std}(P_d)$, so that $P \overset{b[n]}{\rightsquigarrow} R$ cannot be deduced from a rule of type (1R), and must have been derived from $P_d \overset{b[n]}{\rightsquigarrow} R_d$ via rule (2R) with the same d. By induction, either (1) $m = n$, in which case $a = b$, $Q_d = R_d$ and $Q = f(\overrightarrow{Q}) = f(\overrightarrow{R}) = R$, or else (2) $m \neq n$ and there is S_d such that $Q_d \overset{b[n]}{\rightsquigarrow} S_d$ and $R_d \overset{a[m]}{\rightsquigarrow} S_d$. Let $S = f(\overrightarrow{S})$. Then $Q \overset{b[n]}{\rightsquigarrow} S$ and $R \overset{a[m]}{\rightsquigarrow} S$ by rules of type (2R) again.

(3) Suppose $P = f_r[m'](\overrightarrow{P})$ for some m'. Reverse transitions from P can only come from rules of type (3R) or (3′R).

(a) Suppose that $P \overset{a[m]}{\rightsquigarrow} Q$ is derived via rule (3R):

$$\frac{\{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f_r[m](\overrightarrow{X}) \overset{a[m]}{\rightsquigarrow} f(\overrightarrow{X})}$$

In this case we must have $m' = m$. Then $P \overset{b[n]}{\rightsquigarrow} R$ cannot come from a rule of type (3′R), since $\mathsf{std}(P_{\mathsf{ta}(r)})$, so that $P_{\mathsf{ta}(r)} \not\rightarrow$. So $P \overset{b[n]}{\rightsquigarrow} R$ must come from the same rule as $P \overset{a[m]}{\rightsquigarrow} Q$, which implies that $m = n$ and $a = b$.

(b) Suppose that $P \overset{a[m]}{\rightsquigarrow} Q$ is derived from $P_{\mathsf{ta}(r)} \overset{a[m]}{\rightsquigarrow} Q_{\mathsf{ta}(r)}$ via a rule of type (3′R):

$$\frac{X_{\mathsf{ta}(r)} \overset{a[m]}{\rightsquigarrow} X'_{\mathsf{ta}(r)} \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{\mathsf{ta}(r)\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f_r[m'](\overrightarrow{X}) \overset{a[m]}{\rightsquigarrow} f_r[m'](\overrightarrow{X'})}\ m \neq m'$$

Then not $\mathsf{std}(P_{\mathsf{ta}(r)})$ and so $P \overset{b[n]}{\rightsquigarrow} R$ must be derived from $P_{\mathsf{ta}(r)} \overset{b[n]}{\rightsquigarrow} R_{\mathsf{ta}(r)}$ via a rule of type (3′R). By induction, either (1) $m = n$, in which case $a = b$, $Q_{\mathsf{ta}(r)} = R_{\mathsf{ta}(r)}$ and $Q = R$, or else (2) there is $S_{\mathsf{ta}(r)}$ such that $Q_{\mathsf{ta}(r)} \overset{b[n]}{\rightsquigarrow} S_{\mathsf{ta}(r)}$ and $R_{\mathsf{ta}(r)} \overset{a[m]}{\rightsquigarrow} S_{\mathsf{ta}(r)}$. Let $S = f_r[m'](\overrightarrow{S})$. Then it is easy to see that $Q \overset{b[n]}{\rightsquigarrow} S$ and $R \overset{a[m]}{\rightsquigarrow} S$ as required. □

B. Proof of Proposition 5.19

We require a lemma:

Lemma B.1. *Suppose that* $P, Q \in \mathsf{Proc}$ *and* $P = f(\overrightarrow{P}) \rightarrow^* Q$. *Then* Q *is of the form either* $f(\overrightarrow{Q})$ *or* $f_r[n](\overrightarrow{Q})$ *for some* $\overrightarrow{Q}$. *Moreover, if* $m \in \mathsf{keys}(P_i)$ *and* $m \notin \mathsf{keys}(P_j)$ *for some* i, j, *then* $m \in \mathsf{keys}(Q_i)$ *and* $m \notin \mathsf{keys}(Q_j)$.

Proof of Proposition 5.19. We prove both parts at once. We use induction on the size of P, i.e. how many operators occur in P. Note that this is invariant under computation: if $P \xrightarrow{\mu} Q$ then P and Q have the same size.

(1) If $P = \mathbf{0}$ then no transitions are possible and we are done.

(2) Suppose $P = f(\overrightarrow{P})$. Plainly forward transitions from P can only come from rules of type (1), (2) or (3).

(a) Suppose that $P \xrightarrow{a[m]} Q$ is derived from $P_d \xrightarrow{a[m]} Q_d$ via a rule of type (2), with $Q = f(\overrightarrow{Q})$. Then we must have $\mathsf{std}(P_e)$ for all $e \in E$ except d, and not $\mathsf{std}(Q_d)$. Any forward transition from Q onwards must use a rule of type (2). Hence T is of the form $f(\overrightarrow{T})$ where $T_i = P_i$ for all $i \in N \setminus \{d\}$. Also $Q_d \xrightarrow{s} T_d$. We are given $P \xrightarrow{b[n]} R \xrightarrow{t} T$. The transition $P \xrightarrow{b[n]} R$ cannot be derived by a rule of type (1), since this would mean that some P_i for $i \in S$ would be changed in T. Also $P \xrightarrow{b[n]} R$ cannot be derived by a rule of type (3), since then T would have operator $f_r[n]$. So $P \xrightarrow{b[n]} R$ must be derived from a rule of type (2). Also this rule must have active argument d. So we have $P_d \xrightarrow{b[n]} R_d \xrightarrow{t} T_d$. By induction, either (1) $m = n$, in which case $a = b$, $Q_d = R_d$, and so $Q = R$, or (2) $m \neq n$ and there is S_d such that $Q_d \xrightarrow{b[n]} S_d$, $R_d \xrightarrow{a[m]} S_d$, and $S_d \xrightarrow{s \setminus b[n]} T_d$, $S_d \xrightarrow{t \setminus a[m]} T_d$. In case (2) we let $S = f(\overrightarrow{S})$, where $S_i = P_i$ for $i \neq d$, and we see that S is as required.

(b) Suppose that $P \xrightarrow{a[m]} Q$ is derived by a rule of type (1) from $\{P_i \xrightarrow{a_i[m]} Q_i\}_{i \in I}$. We know from item (2a) that $P \xrightarrow{b[n]} R$ cannot be derived by a rule of type (2). If it is derived via a rule of type (3) then for any $i \in I$, the argument P_i remains unchanged along the computation from P to T via R, whereas we know that P_i has been changed in T because of $P \xrightarrow{a[m]} Q$. Therefore $P \xrightarrow{b[n]} R$ must be derived by a rule of type (1), from premises $\{P_i \xrightarrow{b_i[n]} R_i\}_{i \in I'}$.

Suppose $m = n$. We claim that $I = I'$. This follows from Lemma B.1 applied to the computations $Q \xrightarrow{s} T$ and $R \xrightarrow{t} T$. Then by induction we have $a_i = b_i$ and $Q_i = R_i$ for each $i \in I$. Hence $Q = R$ as required. Also $a = b$, since any two rules of type (1) with the same premises are required to have the same conclusion.

Now suppose $m \neq n$. Consider the computations $Q \xrightarrow{s} T$ and $R \xrightarrow{t} T$. Any computation consists of the application of (a) all type (1) rules, or (b) some number of type (1) rules followed by some number of type (2) rules, or (c) some number of type (1) rules followed by a single type (3) rule and some number of type (3′) rules. In the case of type (2) or (3′) rules, a single argument in E is involved. So one can infer the number of applications of the rules from the final state, in this case T. Also the arguments in S are unaffected by type (2), (3) or (3′) rules. If we remove any uses of type (2), (3) or (3′) rules from s and t, we get $Q \xrightarrow{s'} T'$ and $R \xrightarrow{t'} T'$, where s', t' only use type (1) rules. Thus without loss of generality we can assume that s and t involve only type (1) rules. In particular, T is of the form $f(\overrightarrow{T})$.

By Proposition 5.18, $s = s'b[n]s''$ for some s', s''. Also by Proposition 5.18, for $i \in I'$ we have $Q_i \xrightarrow{s_i' b_i[n] s_i''} T_i$. For $i \notin I'$ we have $Q_i \xrightarrow{s_i' s_i''} T_i$. Here the computations s_i' come before the $b[n]$ transition, and the s_i'' after $b[n]$.

Similarly $t = t'a[m]t''$ for some t', t''. For $i \in I$ we have $R_i \xrightarrow{t_i' a_i[m] t_i''} T_i$. For $i \notin I$ we have $R_i \xrightarrow{t_i' t_i''} T_i$.

By induction, for each $i \in I \cap I'$ there is S_i such that $Q_i \xrightarrow{b_i[n]} S_i$, $R_i \xrightarrow{a_i[m]} S_i$. and $S_i \xrightarrow{s_i' s_i''} T_i$, $S_i \xrightarrow{t_i' t_i''} T_i$. For $i \in I \setminus I'$, let $S_i = Q_i$. We have $S_i \xrightarrow{s_i' s_i''} T_i$. For $i \in I' \setminus I$, let $S_i = R_i$. We have $P_i = Q_i \xrightarrow{s_i' b_i[n] s_i''} T_i$ and $P_i \xrightarrow{b_i[n]} R_i \xrightarrow{t_i' t_i''} T_i$. By repeated use of the induction hypothesis on P_i and its successors along the s_i'

computation, we deduce that $R_i = S_i \stackrel{s'_i s''_i}{\to} T_i$. Finally for $i \notin I \cup I'$ let $S_i = P_i$. We have $S_i \stackrel{s'_i s''_i}{\to} T_i$. Thus we have established that for every $i \in N$, $S_i \stackrel{s'_i s''_i}{\to} T_i$.

Let $S = f(\overrightarrow{S})$. We have $Q \stackrel{b[n]}{\to} S$ and $R \stackrel{a[m]}{\to} S$. By combining all the $s'_i s''_i$ computations we have $S \stackrel{s's''}{\to} T$, i.e. $S \stackrel{s\backslash b[n]}{\to} T$ as required. Similarly we can show that $S \stackrel{t\backslash a[m]}{\to} T$.

(c) Suppose that $P \stackrel{a[m]}{\to} Q$ is derived by a rule of type (3). Then by items (2a) and (2b), $P \stackrel{b[n]}{\to} R$ must also be derived by a rule of type (3). But T must be of the form $f_r[m](\overrightarrow{T})$. So R must be derived by exactly the same rule as Q. So $m = n$, $a = b$ and $Q = R$, and we are done.

(3) Suppose $P = f_r[m'](\overrightarrow{P})$ for some m'. Forward transitions from P can only come from rules of type (3′). All derivatives of P must have the form $f_r[m'](\overrightarrow{Q})$ where $Q_i = P_i$ for all $i \in N \setminus \{\mathsf{ta}(r)\}$. By induction the result holds for $P_{\mathsf{ta}(r)}$, and we deduce that the result holds for P. □

C. Proof of Theorem 5.21

(1) By induction on the derivation of $P \stackrel{a[m]}{\to} P'$. There are four cases:

(a) Suppose that $f(\overrightarrow{P}) \stackrel{a[m]}{\to} f(\overrightarrow{P'})$ is deduced from $\{P_i \stackrel{a_i[m]}{\to} P'_i\}_{i\in I}$ via the type (1) rule

$$\frac{\{X_i \stackrel{a_i[m]}{\to} X'_i\}_{i\in I} \quad \{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S\backslash I}}{f(\overrightarrow{X}) \stackrel{a[m]}{\to} f(\overrightarrow{X'})}.$$

By induction we get $\pi(P_i) \stackrel{a_i}{\to}_{\mathrm{S}} \pi(P'_i)$ for all $i \in I$. We deduce $f(\overrightarrow{\pi(P)}) \stackrel{a}{\to}_{\mathrm{S}} f(\overrightarrow{\pi(P')})$ from the type (I) rule

$$\frac{\{X_i \stackrel{a_i}{\to}_{\mathrm{S}} X'_i\}_{i\in I}}{f(\overrightarrow{X}) \stackrel{a}{\to}_{\mathrm{S}} f(\overrightarrow{X'})}.$$

But $\pi(f(\overrightarrow{P})) = f(\overrightarrow{\pi(P)})$ and $\pi(f(\overrightarrow{P'})) = f(\overrightarrow{\pi(P')})$, since $P_d \in \mathsf{Std}$ for all $d \in D_f$. Hence $\pi(f(\overrightarrow{P})) \stackrel{a}{\to}_{\mathrm{S}} \pi(f(\overrightarrow{P'}))$ as required.

(b) Suppose that $f(\overrightarrow{P}) \stackrel{a[m]}{\to} f(\overrightarrow{P'})$ is deduced from $P_d \stackrel{a[m]}{\to} P'_d$ via the type (2) rule

$$\frac{X_d \stackrel{a[m]}{\to} X'_d \quad \{\mathsf{std}(X_e)\}_{e\in E\backslash\{d\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f(\overrightarrow{X}) \stackrel{a[m]}{\to} f(\overrightarrow{X'})}$$

for some $d \in D_f$. Then by induction we know that $\pi(P_d) \stackrel{a}{\to}_{\mathrm{S}} \pi(P'_d)$. Also by Lemma 5.2 we have $P'_d \notin \mathsf{Std}$. Hence $\pi(f(\overrightarrow{P'})) = \pi(P'_d)$.

If $P_d \in \mathsf{Std}$ then $\pi(f(\overrightarrow{P})) = f(\overrightarrow{\pi(P)})$. By the type (II) rule

$$\frac{X_d \stackrel{a}{\to}_{\mathrm{S}} X'_d}{f(\overrightarrow{X}) \stackrel{a}{\to}_{\mathrm{S}} X'_d}$$

we deduce $f(\overrightarrow{\pi(P)}) \stackrel{a}{\to}_{\mathrm{S}} \pi(P'_d)$. So $\pi(f(\overrightarrow{P})) \stackrel{a}{\to}_{\mathrm{S}} \pi(f(\overrightarrow{P'}))$.

If $P_d \notin \mathsf{Std}$ then $\pi(f(\overrightarrow{P})) = \pi(P_d)$. Again we get $\pi(f(\overrightarrow{P})) \stackrel{a}{\to}_{\mathrm{S}} \pi(f(\overrightarrow{P'}))$, as required.

(c) Suppose that $f(\overrightarrow{P}) \stackrel{a[m]}{\to} f_r[m](\overrightarrow{P})$ is deduced from the type (3) rule

$$\frac{\{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f(\overrightarrow{X}) \stackrel{a[m]}{\to} f_r[m](\overrightarrow{X})}.$$

Then $\pi(f(\overrightarrow{P})) = f(\overrightarrow{\pi(P)})$ since $P_d \in \mathsf{Std}$ for all $d \in D_f$. Also we obtain $\pi(f_r[m](\overrightarrow{P})) = \pi(P_{\mathsf{ta}(r)})$. We can deduce $f(\overrightarrow{\pi(P)}) \stackrel{a}{\to}_{\mathrm{S}} P_{\mathsf{ta}(r)}$ from type (III) rule

r $\dfrac{}{f(\overrightarrow{X}) \xrightarrow{a}_{\mathrm{S}} X_{\mathsf{ta}(r)}}$.

Hence $\pi(f(\overrightarrow{P})) \xrightarrow{a}_{\mathrm{S}} \pi(f_r[m](\overrightarrow{P}))$ as required.

(d) Suppose that $f_r[n](\overrightarrow{P}) \xrightarrow{a[m]} f_r[n](\overrightarrow{P'})$ is deduced from $P_{\mathsf{ta}(r)} \xrightarrow{a[m]} P'_{\mathsf{ta}(r)}$ via the type (3′) rule

$$\frac{X_{\mathsf{ta}(r)} \xrightarrow{a[m]} X'_{\mathsf{ta}(r)} \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{\mathsf{ta}(r)\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f_r[n](\overrightarrow{X}) \xrightarrow{a[m]} f_r[n](\overrightarrow{X'})} \quad m \neq n$$

By induction we have $\pi(P_{\mathsf{ta}(r)}) \xrightarrow{a} \pi(P'_{\mathsf{ta}(r)})$. Also $\pi(f_r[n](\overrightarrow{P})) = \pi(P_{\mathsf{ta}(r)})$ and $\pi(f_r[n](\overrightarrow{P'})) = \pi(P'_{\mathsf{ta}(r)})$. Hence we obtain $\pi(f_r[n](\overrightarrow{P})) \xrightarrow{a}_{\mathrm{S}} \pi(f_r[n](\overrightarrow{P'}))$ as required.

(2) By induction on the definition of $\pi(P)$ (structural induction on P). Suppose $m \in \mathcal{K} \setminus \mathsf{keys}(P)$.

(a) Suppose $P = \mathbf{0}$. Then $\pi(P) = \mathbf{0}$ and there is nothing to prove.

(b) Suppose $P = f(\overrightarrow{P})$. There are three cases:

(i) Suppose that for some $d \in D_f$ we have $P_d \notin \mathsf{Std}$, with $P_e \in \mathsf{Std}$ for all $e \in E_f \setminus \{d\}$. Then $\pi(P) = \pi(P_d)$. Suppose that $\pi(P_d) \xrightarrow{a}_{\mathrm{S}} P'$. By induction there is P'_d such that $P_d \xrightarrow{a[m]} P'_d$ and $\pi(P'_d) = P'$ (note that $m \notin \mathsf{keys}(P_d)$). But then $f(\overrightarrow{P}) \xrightarrow{a[m]} f(\overrightarrow{P'})$ via the type (2) rule

$$\frac{X_d \xrightarrow{a[m]} X'_d \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{d\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f(\overrightarrow{X}) \xrightarrow{a[m]} f(\overrightarrow{X'})}.$$

Also $\pi(f(\overrightarrow{P'})) = \pi(P'_d) = P'$. So $f(\overrightarrow{P'})$ is the required P''.

(ii) Suppose that $P_e \in \mathsf{Std}$ for all $e \in E_f$. Then $\pi(P) = f(\overrightarrow{\pi(P)})$. Suppose $\pi(P) \xrightarrow{a}_{\mathrm{S}} P'$. There are three subcases, depending on which rule is used to deduce the transition.

(A) Static rule. We have $\pi(P_i) \xrightarrow{a_i}_{\mathrm{S}} P'_i$ for $i \in I$, and $f(\overrightarrow{\pi(P)}) \xrightarrow{a}_{\mathrm{S}} P' = f(\overrightarrow{P'})$ is deduced via the type (I) rule

$$\frac{\{X_i \xrightarrow{a_i}_{\mathrm{S}} X'_i\}_{i\in I}}{f(\overrightarrow{X}) \xrightarrow{a}_{\mathrm{S}} f(\overrightarrow{X'})}.$$

Here we let $P'_j = \pi(P_j)$ for $j \notin I$. By induction, for each $i \in I$ there is P''_i such that $P_i \xrightarrow{a_i[m]} P''_i$ and $\pi(P''_i) = P'_i$. We deduce $f(\overrightarrow{P}) \xrightarrow{a[m]} f(\overrightarrow{P''})$ via the type (1) rule

$$\frac{\{X_i \xrightarrow{a_i[m]} X'_i\}_{i\in I} \quad \{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S\setminus I}}{f(\overrightarrow{X}) \xrightarrow{a[m]} f(\overrightarrow{X'})}.$$

Here we let $P''_j = P_j$ for $j \notin I$. Now $\pi(f(\overrightarrow{P''})) = f(\overrightarrow{\pi(P'')}) = f(\overrightarrow{P'})$ and we are done.

(B) Choice rule. There is $d \in D_f$ such that $f(\overrightarrow{\pi(P)}) \xrightarrow{a}_{\mathrm{S}} P'$ is deduced from $\pi(P_d) \xrightarrow{a}_{\mathrm{S}} P'$ via the type (II) rule

$$\frac{X_d \xrightarrow{a}_{\mathrm{S}} X'_d}{f(\overrightarrow{X}) \xrightarrow{a}_{\mathrm{S}} X'_d}.$$

By induction, there is P'_d such that $P_d \xrightarrow{a[m]} P'_d$ and $\pi(P'_d) = P'$. But then we can deduce $f(\overrightarrow{P}) \xrightarrow{a[m]} f(\overrightarrow{P'})$ via the type (2) rule

$$\frac{X_d \xrightarrow{a[m]} X'_d \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{d\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f(\overrightarrow{X}) \xrightarrow{a[m]} f(\overrightarrow{X'})}.$$

Also $P'_d \notin \mathsf{Std}$ by Lemma 5.2, and so $\pi(f(\overrightarrow{P'})) = \pi(P'_d) = P'$.

(C) Choice axiom. Suppose $f(\overrightarrow{\pi(P)}) \xrightarrow{a}_{\mathrm{S}} \pi(P_{\mathsf{ta}(r)})$ is deduced via the type (III) rule

$$r \quad \frac{}{f(\overrightarrow{X}) \xrightarrow{a}_{\mathrm{S}} X_{\mathsf{ta}(r)}}\ .$$

Then we have $f(\overrightarrow{P}) \xrightarrow{a[m]} f_r[m](\overrightarrow{P})$ via the type (3) rule

$$\frac{\{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f(\overrightarrow{X}) \xrightarrow{a[m]} f_r[m](\overrightarrow{X})}\ .$$

Also $\pi(f_r[m](\overrightarrow{P})) = \pi(P_{\mathsf{ta}(r)})$ and we are done.

(iii) Otherwise, $\pi(P) = \mathbf{0}$ and there is nothing to prove.

(c) Suppose $P = f_r[n](\overrightarrow{P})$ where r is a choice axiom with operator f. Note that $n \neq m$.

If $P_e \in \mathsf{Std}$ for all $e \in E_f \setminus \{\mathsf{ta}(r)\}$ then $\pi(P) = \pi(P_{\mathsf{ta}(r)})$. Suppose that $\pi(P_{\mathsf{ta}(r)}) \xrightarrow{a}_{\mathrm{S}} P'$. By induction there is $P'_{\mathsf{ta}(r)}$ such that $P_{\mathsf{ta}(r)} \xrightarrow{a[m]} P'_{\mathsf{ta}(r)}$ and $\pi(P'_{\mathsf{ta}(r)}) = P'$. But then $f_r[n](\overrightarrow{P}) \xrightarrow{a[m]} f_r[n](\overrightarrow{P'})$ via the type (3′) rule

$$\frac{X_{\mathsf{ta}(r)} \xrightarrow{a[m]} X'_{\mathsf{ta}(r)} \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{\mathsf{ta}(r)\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f_r[n](\overrightarrow{X}) \xrightarrow{a[m]} f_r[n](\overrightarrow{X'})} \quad m \neq n$$

Also $\pi(f_r[n](\overrightarrow{P'})) = \pi(P'_{\mathsf{ta}(r)}) = P'$.

If $P_e \notin \mathsf{Std}$ for some $e \in E_f \setminus \{\mathsf{ta}(r)\}$ then $\pi(P) = \mathbf{0}$ and there is nothing to prove. □

D. Proof of Theorem 6.7

Let $L = (\Sigma_{\mathrm{SA}}, \mathsf{ActK}, R_{\mathrm{F}}, R_{\mathrm{R}})$ be a reversible process calculus with communication keys, and Proc, $\rightarrow$ and $\rightsquigarrow$ be generated by L as in Section 3. We start by defining a relation $\mathcal{R} \subseteq \mathsf{Proc} \times \mathsf{Proc}$ as the least relation that satisfies $\overrightarrow{P}\,\mathcal{R}\,\overrightarrow{Q}$ if $\overrightarrow{P} \sim_{\mathsf{FR}} \overrightarrow{Q}$, and $f(\overrightarrow{P})\mathcal{R}f(\overrightarrow{Q})$ if $\overrightarrow{P}\,\mathcal{R}\,\overrightarrow{Q}$, where f is either a member of Σ_{S} or is an auxiliary operator in Σ_{A}. Hence, $\mathcal{R}$ is the least congruence which contains $\sim_{\mathsf{FR}}$.

All we need to do next is to show by structural induction that $\mathcal{R}$ is a forward–reverse bisimulation by verifying the properties in Definition 6.5 for all appropriate terms $f(\overrightarrow{P})$ and $f(\overrightarrow{P})$ such that $f(\overrightarrow{P})\mathcal{R}f(\overrightarrow{Q})$. Either $f(\overrightarrow{P}) \sim_{\mathsf{FR}} f(\overrightarrow{P})$ and we are done, or we deduce $\overrightarrow{P}\,\mathcal{R}\,\overrightarrow{Q}$. By inductive hypothesis all the corresponding elements P_i and Q_i satisfy the properties of Definition 6.5. Below, we verify the three properties of forward–reverse bisimulation for $f(\overrightarrow{P})$ and $f(\overrightarrow{P})$.

Property 1. We only have two predicates std and $\mathsf{fsh}[m]$ to consider:

Assume $\mathsf{std}(f(\overrightarrow{P}))$. By the rules for std in Section 3 we deduce $\{\mathsf{std}(P_i)\}_{i\in N_f}$. This is equivalent to $\{\mathsf{std}(Q_i)\}_{i\in N_f}$ by induction, hence $\mathsf{std}(f(\overrightarrow{Q}))$ by the second rule for std. The converse follows correspondingly.

For the predicate $\mathsf{fsh}[m]$ we proceed similarly. Let f be some $f_r \in \Sigma_A$. From $\mathsf{fsh}[m](f_r[n](\overrightarrow{P}))$, where $n \neq m$, we deduce $\mathsf{fsh}[m](P_i)$ for all $i \in N_{f_r}$ by the third rule for $\mathsf{fsh}[m]$ in Section 3. Then by induction we obtain $\mathsf{fsh}[m](Q_i)$ for all appropriate i, and $\mathsf{fsh}[m](f_r[n](\overrightarrow{Q}))$ by the same rule. The case where $f \in \Sigma_{\mathrm{S}}$ is proved correspondingly, as is the converse.

Property 2. There are four cases due to rules of type (1), (2), (3) and (3′):

(1) Suppose that $f(\overrightarrow{P}) \xrightarrow{a[m]} f(\overrightarrow{P'})$ is deduced from $\{P_i \xrightarrow{a_i[m]} P'_i\}_{i\in I}$ via the following rule:

$$\frac{\{X_i \xrightarrow{a_i[m]} X'_i\}_{i\in I} \quad \{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S\setminus I}}{f(\overrightarrow{X}) \xrightarrow{a[m]} f(\overrightarrow{X'})}$$

Note that I and E are disjoint. We also must have $\{\mathsf{std}(P_e)\}_{e\in E}$ and $\{\mathsf{fsh}[m](P_i)\}_{i\in S\setminus I}$.

By induction we get $Q_i \xrightarrow{a_i[m]} Q'_i$ for all $i \in I$, and also $\{\mathsf{std}(Q_e)\}_{e\in E}$ and $\{\mathsf{fsh}[m](Q_i)\}_{i\in Si\setminus I}$. So, the rule above proves $f(\overrightarrow{Q}) \xrightarrow{a[m]} f(\overrightarrow{Q'})$ and since $\overrightarrow{P'}\mathcal{R}\overrightarrow{Q'}$ we deduce $f(\overrightarrow{P'})\mathcal{R}f(\overrightarrow{Q'})$.

(2) Suppose that $f(\overrightarrow{P}) \overset{a[m]}{\to} f(\overrightarrow{P'})$ is deduced from $P_d \overset{a[m]}{\to} P'_d$ via the rule

$$\frac{X_d \overset{a[m]}{\to} X'_d \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{d\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f(\overrightarrow{X}) \overset{a[m]}{\to} f(\overrightarrow{X'})}$$

for some $d \in D_f$. We also must have $\mathsf{std}(P_e)\}_{e\in E\setminus\{d\}}$ and $\mathsf{fsh}[m](P_i)\}_{i\in S}$. Note that $d \notin S$. Then by induction we know that $Q_d \overset{a[m]}{\to} Q'_d$ and $P'_d \mathcal{R} Q'_d$. Also, we deduce $\mathsf{std}(Q_e)\}_{e\in E\setminus\{d\}}$ and $\mathsf{fsh}[m](Q_i)\}_{i\in S}$. Hence, by the last rule we obtain $f(\overrightarrow{Q}) \overset{a[m]}{\to} f(\overrightarrow{Q'})$ where $Q'_i = Q_i$ for all $i \neq d$. Since $\overrightarrow{P'} \mathcal{R} \overrightarrow{Q'}$ we know that $f(\overrightarrow{P'}) \mathcal{R} f(\overrightarrow{Q'})$.

(3) Suppose that $f(\overrightarrow{P}) \overset{a[m]}{\to} f_r[m](\overrightarrow{P})$ is deduced by the rule

$$\frac{\{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f(\overrightarrow{X}) \overset{a[m]}{\to} f_r[m](\overrightarrow{X})}$$

This is due to $\{\mathsf{std}(P_e)\}_{e\in E}$ and $\{\mathsf{fsh}[m](P_i)\}_{i\in S}$. Since the corresponding processes in $\overrightarrow{P}$ and $\overrightarrow{Q}$ satisfy the first property of Definition 6.5, we know that $\{\mathsf{std}(Q_e)\}_{e\in E}$ and $\{\mathsf{fsh}[m](Q_i)\}_{i\in S}$. Then by the above rule $f(\overrightarrow{Q}) \overset{a[m]}{\to} f_r[m](\overrightarrow{Q})$, and $f_r[m](\overrightarrow{P}) \mathcal{R} f_r[m](\overrightarrow{Q})$ by the congruence property of $\mathcal{R}$.

(4) Suppose that $f_r[n](\overrightarrow{P}) \overset{a[m]}{\to} f_r[n](\overrightarrow{P'})$ is deduced from $P_{\mathsf{ta}(r)} \overset{a[m]}{\to} P'_{\mathsf{ta}(r)}$ via the rule

$$\frac{X_{\mathsf{ta}(r)} \overset{a[m]}{\to} X'_{\mathsf{ta}(r)} \quad \{\mathsf{std}(X_e)\}_{e\in E\setminus\{\mathsf{ta}(r)\}} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f_r[n](\overrightarrow{X}) \overset{a[m]}{\to} f_r[n](\overrightarrow{X'})} \quad m \neq n$$

Since $\mathsf{ta}(r) \notin S$ this case is proved in the same way as the first case.

Property 3. We proceed very much in the same way as for Property 2 due to the symmetry between the corresponding forward and reverse rules. We shall only consider reverse transitions deduced from reverse rules (2R) and (3R); transitions proved from reverse rules (1R) and (3′R) are dealt with correspondingly.

(1) Assume that $f(\overrightarrow{P}) \overset{a[m]}{\rightsquigarrow} f(\overrightarrow{P'})$ is derived from $\{P_i \overset{a_i[m]}{\rightsquigarrow} P'_i\}_{i\in I}$ via the following rule of type (2R):

$$\frac{\{X_i \overset{a_i[m]}{\rightsquigarrow} X'_i\}_{i\in I} \quad \{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S\setminus I}}{f(\overrightarrow{X}) \overset{a[m]}{\rightsquigarrow} f(\overrightarrow{X'})}$$

This also requires $\{\mathsf{std}(P_e)\}_{e\in E}$ and $\{\mathsf{fsh}[m](P_i)\}_{i\in S\setminus I}$.

By induction we get $Q_i \overset{a_i[m]}{\rightsquigarrow} Q'_i$ for all $i \in I$, and since $I \cap E = \emptyset$ we obtain also $\{\mathsf{std}(Q_e)\}_{e\in E}$ and $\{\mathsf{fsh}[m](Q_i)\}_{i\in S\setminus I}$. Moreover, $\overrightarrow{P'} \mathcal{R} \overrightarrow{Q'}$. So, the last rule proves $f(\overrightarrow{Q}) \overset{a[m]}{\to} f(\overrightarrow{Q'})$ and we deduce $f(\overrightarrow{P'}) \mathcal{R} f(\overrightarrow{Q'})$ by the congruence property of $\mathcal{R}$.

(2) Assume that $f_r[m](\overrightarrow{P}) \overset{a[m]}{\rightsquigarrow} f(\overrightarrow{P})$ is deduced by the rule of type (3′R)

$$\frac{\{\mathsf{std}(X_e)\}_{e\in E} \quad \{\mathsf{fsh}[m](X_i)\}_{i\in S}}{f_r[m](\overrightarrow{X}) \overset{a[m]}{\rightsquigarrow} f(\overrightarrow{X})}$$

from $\{\mathsf{std}(P_e)\}_{e\in E}$ and $\{\mathsf{fsh}[m](P_i)\}_{i\in S}$. Since the corresponding elements of $\overrightarrow{P}$ and $\overrightarrow{Q}$ satisfy the first property of forward–reverse bisimulation, we know that $\{\mathsf{std}(Q_e)\}_{e\in E}$ and $\{\mathsf{fsh}[m](Q_i)\}_{i\in S}$. Then by the above rule $f_r[m](\overrightarrow{Q}) \overset{a[m]}{\rightsquigarrow} f(\overrightarrow{Q})$, and $f(\overrightarrow{P}) \mathcal{R} f(\overrightarrow{Q})$. □

References

[1] S. Abramsky, A structural approach to reversible computation, Theoret. Comput. Sci. 347 (3) (2005) 441–464.

[2] T. Altenkirch, J. Grattage, A functional quantum programming language, Proceedings of 20th Annual IEEE Symposium on Logic in Computer Science, LICS 2005, IEEE Computer Society Press, 2005, pp. 249–258.

[3] J.C.M. Baeten, C. Verhoef, A congruence theorem for structured operational semantics with predicates, Proceedings of 4th International Conference on Concurrency Theory, CONCUR 1993, Lecture Notes in Computer Science, vol. 715, Springer-Verlag, 1993, pp. 477–492.

[4] J.C.M. Baeten, W.P. Weijland, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1990.
[5] M.A. Bednarczyk, Hereditary history preserving bisimulations or what is the power of the future perfect in program logics, Technical Report, Institute of Computer Science, Polish Academy of Sciences, Gdańsk, 1991.
[6] G. Boudol, I. Castellani, A non-interleaving semantics for CCS based on proved transitions, Fund. Inform. 11 (4) (1988) 433–452.
[7] G. Boudol, I. Castellani, Flow models of distributed computations: three equivalent semantics for CCS, Inform. Comput. 114 (2) (1994) 247–314.
[8] H. Buhrman, J. Tromp, P. Vitányi, Time and space bounds for reversible simulation, Proceedings of 28th International Colloquium on Automata, Languages and Programming, ICALP 2001, Lecture Notes in Computer Science, vol. 2076, Springer-Verlag, 2001, pp. 1017–1027.
[9] I. Castellani, Process algebras with localities, in: J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, Elsevier Science, 2001, pp. 945–1045.
[10] V. Danos, J. Krivine, Formal molecular biology done in CCS-R, in: Proceedings of Workshop on Concurrent Models in Molecular Biology, BioConcur 2003, Marseille, September 2003, 2003.
[11] V. Danos, J. Krivine, Reversible communicating systems, Proceedings of 15th International Conference on Concurrency Theory, CONCUR 2004, Lecture Notes in Computer Science, vol. 3170, Springer-Verlag, 2004, pp. 292–307.
[12] V. Danos, J. Krivine, Transactions in RCCS, Proceedings of 16th International Conference on Concurrency Theory, CONCUR 2005, Lecture Notes in Computer Science, vol. 3653, Springer-Verlag, 2005, pp. 398–412.
[13] R. De Nicola, U. Montanari, F. Vaandrager, Back and forth bisimulations, Proceedings of CONCUR'90, Theories of Concurrency: Unification and Extension, Lecture Notes in Computer Science, vol. 458, Springer-Verlag, 1990, pp. 152–165.
[14] P. Degano, R. De Nicola, U. Montanari, Partial ordering derivations for CCS, Proceedings of Fundamentals of Computation Theory, FCT'85, Lecture Notes in Computer Science, vol. 199, Springer-Verlag, 1985, pp. 520–533.
[15] P. Degano, R. De Nicola, U. Montanari, A partial ordering semantics for CCS, Theoret. Comput. Sci. 75 (1990) 223–262.
[16] P. Degano, C. Priami, Enhanced operational semantics: a tool for describing and analyzing concurrent systems, ACM Comput. Surveys 33 (2) (2001) 135–176.
[17] U. Goltz, R. Kuiper, W. Penczek, Propositional temporal logics and equivalences, Proceedings of 3rd International Conference on Concurrency Theory, CONCUR 1992, Lecture Notes in Computer Science, vol. 630, Springer-Verlag, 1992, pp. 222–235.
[18] J.F. Groote, F.W. Vaandrager, Structured operational semantics and bisimulation as a congruence, Inform. Comput. 100 (1992) 202–260.
[19] R.J. van Glabbeek, History preserving process graphs, Draft 20 June 1996. <http://boole.stanford.edu/∼rvg/pub/history.draft.dvi, 1996>.
[20] R.J. van Glabbeek, U. Goltz, Refinement of actions and equivalence notions for concurrent systems, Acta Inform. 37 (4/5) (2001) 229–327.
[21] C.A.R. Hoare, Communicating sequential processes, Commun. Assoc. Comput. Mach. 21 (8) (1978) 666–677.
[22] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.
[23] R. Landauer, Irreversibility and heat generated in the computing process, IBM J. Res. Develop. 5 (1961) 183–191.
[24] R. Milner, Communication and Concurrency, Prentice-Hall, 1989.
[25] R. Milner, Functions as processes, Math. Structures Comput. Sci. 2 (1992) 119–141.
[26] M.R. Mousavi, M.A. Reniers, Congruence for structural congruences, Proceedings of 8th International Conference on Foundations of Software Science and Computation Structures, FOSSACS 2005, Lecture Notes in Computer Science, vol. 3441, Springer-Verlag, 2005, pp. 47–62.
[27] I.C.C. Phillips, I. Ulidowski, Reversing algebraic process calculi, Proceedings of 9th International Conference on Foundations of Software Science and Computation Structures, FOSSACS 2006, Lecture Notes in Computer Science, vol. 3921, Springer-Verlag, 2006, pp. 246–260.
[28] G.D. Plotkin, A structural approach to operational semantics, J. Log. Algebr. Program. 60–61 (2004) 17–139.
[29] Virtutech, Simics Hindsight, 2005. <http://www.virtutech.com>.